Filip Konior sprawozdanie nr5

# Basic Class  
from functools import total\_ordering  
  
  
class StanfordCourse:  
 def \_\_init\_\_(self, department, code, title, students={}):  
 self.department = department  
 self.code = code  
 self.title = title  
 self.students = students  
  
 def mark\_attendance(self, \*students):  
 for a in students:  
 self.students[a] = "Present"  
  
 def is\_present(self, student):  
 key, value = student, "Present"  
 return key in self.students and value == self.students[key]  
  
  
stanford\_python = StanfordCourse("CS", "41", "hap.py code: The python programming language")  
  
print(stanford\_python.title)  
print(stanford\_python.code)  
  
#  
print()  
  
  
# Inheritance  
@total\_ordering  
class StanfordCSCourse(StanfordCourse):  
 def \_\_init\_\_(self, department, code, title, students={}, recorded=False):  
 super().\_\_init\_\_(department, code, title, students)  
 self.is\_recorded = recorded  
  
 def \_\_le\_\_(self, other):  
 return (self.code <= other.code)  
  
 def \_\_lt\_\_(self, other):  
 return (self.code < other.code)  
  
 def \_\_eq\_\_(self, other):  
 return (self.code == other.code)  
  
 def \_\_gt\_\_(self, other):  
 return (self.code > other.code)  
  
 def \_\_ne\_\_(self, other):  
 return (self.code != other.code)  
  
 def \_\_ge\_\_(self, other):  
 return (self.code >= other.code)  
  
  
a = StanfordCourse("CS", "106A", "Programming Methodology")  
b = StanfordCourse("CS", "106A", "Programming Abstractions")  
x = StanfordCSCourse("CS", "106A", "Programming Abstractions", recorded=True)  
  
print(a.code)  
print(b.code)  
print(x.is\_recorded)  
  
print(type(a))  
print(isinstance(a, StanfordCourse)) # True  
print(isinstance(b, StanfordCourse)) # True  
print(isinstance(x, StanfordCourse)) # True  
print(isinstance(x, StanfordCSCourse)) # True  
print(type(a) == type(b)) # True  
print(type(b) == type(x)) # False  
print(a == b) # False  
print(b == x) # False  
  
# Additional Attributes ^  
c = StanfordCourse("CS", "106A", "Programming Methodology", {"Adam": "Present", "Daniel": "Not present"})  
print("Is Daniel present? ", c.is\_present("Daniel"))  
c.mark\_attendance("Daniel")  
print("Is Daniel present? ", c.is\_present("Daniel"))  
  
#  
print()  
# Implementing Prerequisites  
cs106a = StanfordCourse("CS", "106A", "Programming Methodology")  
cs106b = StanfordCSCourse("CS", "106B", "Programming Abstractions")  
cs107 = StanfordCSCourse("CS", "107", "Computer Organzation and Systems")  
cs110 = StanfordCSCourse("CS", "110", "Principles of Computer Systems")  
print(cs110 > cs106b) # True  
print(cs107 > cs110) # False  
  
#  
print()  
# Sorting  
  
courses = [cs110, cs106a, cs107, cs106b]  
for x in courses: print(x.code, end=" ") # Not sorted  
courses.sort()  
print()  
for x in courses: print(x.code, end=" ") # => [cs106a, cs106b, cs107, cs110]  
  
#  
print()  
  
  
# SimpleGraph  
  
# Vertex class  
class Vertex:  
 def \_\_init\_\_(self, name="", edges=set()):  
 self.name = name  
 self.edges = edges  
  
  
class Edge:  
 def \_\_init(self, start: Vertex, end: Vertex, cost=1.0, visited=False):  
 self.start = start  
 self.end = end  
 self.cost = cost  
 self.visited = visited  
  
  
class SimpleGraph:  
 def \_\_init\_\_(self, verts=[], edges=[]):  
 self.verts = verts  
 self.edges = edges  
  
 def add\_vertex(v):  
 pass  
  
 def add\_edge(v\_1, v\_2):  
 pass  
  
  
vertex1 = Vertex("First Vertex")  
vertex2 = Vertex("Second Vertex")  
edge1 = Edge()  
  
#  
print()  
#Inheritance  
"""Examples of Single Inheritance"""  
class Transportation:  
 wheels = 0  
  
 def \_\_init\_\_(self):  
 self.wheels = -1  
  
 def travel\_one(self):  
 print("Travelling on generic transportation")  
  
 def travel(self, distance):  
 for \_ in range(distance):  
 self.travel\_one()  
  
 def is\_auto(self):  
 return (self.wheels)# == 4  
  
class Bike(Transportation):  
  
 def travel\_one(self):  
 print("Biking one mile")  
  
class Car(Transportation):  
 wheels = 4  
 # def \_\_init\_\_(self):  
 # self.wheels = 4  
  
  
 def travel\_one(self):  
 print("Driving one mile")  
  
 def make\_sound(self):  
 print("VROOM")  
  
class Ferrari(Car):  
 pass  
  
t = Transportation()  
b = Bike()  
c = Car()  
f = Ferrari()  
  
print(isinstance(t, Transportation)) # True  
  
print(isinstance(b, Bike)) # True  
print(isinstance(b, Transportation)) #True  
print(isinstance(b, Car)) # False  
#print(isinstance(b, t)) # ?  
  
print(isinstance(c, Car)) # True  
print(isinstance(c, Transportation)) # True  
  
print(isinstance(f, Ferrari)) # True  
print(isinstance(f, Car)) # True  
print(isinstance(f, Transportation)) # True  
  
print(issubclass(Bike, Transportation)) # True  
print(issubclass(Car, Transportation)) # True  
print(issubclass(Ferrari, Car)) # True  
print(issubclass(Ferrari, Transportation)) # True  
print(issubclass(Transportation, Transportation)) # True  
  
b.travel(5) # 5x print Biking one mile  
print(c.is\_auto()) # False because wheels = 4 is not in init  
print(f.is\_auto()) # ^  
print(b.is\_auto()) # False  
#print(b.make\_sound())# Error  
c.travel(10) # 10x print driving 1mile  
f.travel(4) # 4x print driving 1mile  
  
#Exceptions  
  
class OutOfRangeError(ValueError):  
 pass  
  
def get\_age():  
 try:  
 a = int(input("How old are you? "))  
 if (a < 0 or a > 123):  
 myError = OutOfRangeError()  
 raise myError  
 print(a)  
 except myError:  
 print(a, ' out of range')  
 except Exception:  
 print("Invalid integer input.")  
 else:  
 print("In else.")  
 finally:  
 print("In finally:")  
  
get\_age()