***PYTHON***

Polymorphism

"""

class Employee:

def \_\_init\_\_(self,rate):

self.rate=rate

def printSalaryPerMonth(self):

print(str(self.rate))

class HourlyEmployee(Employee):

def \_\_init\_\_(self, rate):

super().\_\_init\_\_(rate)

def printSalaryPerMonth(self):

print(str(self.rate\*20\*8))

class SalariedEmployee(Employee):

def \_\_init\_\_(self, rate):

super().\_\_init\_\_(rate)

def printSalaryPerMonth(self):

print(str(self.rate/12))

def PrintWage(emp):

emp.printSalaryPerMonth()

f=HourlyEmployee(10)

g=SalariedEmployee(24000)

f.printSalaryPerMonth() #1600

g.printSalaryPerMonth() #2000

PrintWage(f) #1600

"""

class HourlyEmployee():

def \_\_init\_\_(self, rate):

self.rate=rate

def printSalaryPerMonth(self):

print(str(self.rate\*20\*8))

class SalariedEmployee():

def \_\_init\_\_(self, rate):

self.rate=rate

def printSalaryPerMonth(self):

print(str(self.rate/12))

def PrintWage(emp):

emp.printSalaryPerMonth()

f=HourlyEmployee(10)

g=SalariedEmployee(60000)

f.printSalaryPerMonth() #1600

g.printSalaryPerMonth() #5000

PrintWage(g) #1600

class Day(object):

def \_\_init\_\_(self, visits, contacts):

self.visits = visits

self.contacts = contacts

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

return "Visits: %i, Contacts: %i" % (self.visits, self.contacts)

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

total\_visits = self.visits + other.visits

total\_contacts = self.contacts + other.contacts

return Day(total\_visits, total\_contacts)

day1 = Day(10,1)

day2 = Day(20, 2)

day3 = day1 + day2

print(day3)

Inheritance

class Employee:

def \_\_init\_\_(self,homeAddress):

self.homeAddress=homeAddress

def printAddress(self):

print(self.homeAddress)

class HourlyEmployee(Employee):

def \_\_init\_\_(self,homeAddress,rate):

super().\_\_init\_\_(homeAddress)

self.rate=rate

def printSalaryPerMonth(self):

print("$", str(self.rate\*20\*8))

e=Employee("Henrico")

e.printAddress()

f=HourlyEmployee("Midlothian",10)

f.printSalaryPerMonth()

f.printAddress()

#class A: x = 'a'

#class B(A): pass

#class C(A): x = 'c'

#class D(B, C): pass

#print(D.x) # prints c

'''

class A:

x = 'a'

print("A", x)

class B(A):

print("B")

class C(A):

x = 'c'

print("C", x)

class D(B, C):

print("D")

print(B.x)

print(A.x)

print(D.x)

'''

class A:

def \_\_init\_\_(self):

self.x = 'a'

print("A", self.x)

class B(A):

def \_\_init\_\_(self):

super().\_\_init\_\_()

#self.x = 'b'

print("B", self.x)

class C(A):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.x = 'c'

print("C", self.x)

class D(B, C):

def \_\_init\_\_(self):

super().\_\_init\_\_()

print("D", self.x)

D()

Geometry

import math

class Point:

def \_\_init\_\_(self, x, y): #constructor, 2 arguments

self.x = x

self.y = y

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

return "I'm a Point at (%d, %d)" % (self.x, self.y)

def distance\_from\_origin (self):

return math.sqrt(self.x\*\*2 + self.y\*\*2)

def distance(self, p2):

return math.sqrt((p2.x - self.x)\*\*2 + (p2.y - self.y)\*\*2)

def getOrigin():

return Point(0,0)

pt = Point(10,2)

print(pt) #I'm a point at (10, 2)

print("Distance is: %d" % pt.distance(Point(20,2))) #10.0

Dog

class Dog:

kind = 'canine' # class variable shared by all instances

def \_\_init\_\_(self, name): # constructor, with one argument: name

self.name = name # instance variable unique to each instance

d = Dog('Duffy')

e = Dog("Emo")

print(d.name, "is ", d.kind) # Duffy is canine

print(e.name, "is ", e.kind) # Emo is canine

e.kind = "cat"

print(d.name, "is", d.kind) # Duffy is canine

print(e.name, "is", e.kind) # Emo is cat

def funct(self, x):

print(self.name, x)

d = Dog("Duffy")

#Dog.funct = funct

#d.funct("abc")

print(funct.\_\_get\_\_(d, d.\_\_class\_\_))

d.f = funct.\_\_get\_\_(d, d.\_\_class\_\_)

d.f("efg")

Demos

def minmax(x,y):

if (x<y):

return x,y

else:

return y,x

x,y = 10,5

minXY,maxXY=minmax(x,y)

print(minXY) # 5

print(maxXY) #10

minmaxReturn=minmax(y,x)

print(minmaxReturn[0]) #5

print(minmaxReturn[1]) #10

class Shark:

animal\_type = "fish" #class variable

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

self.name = name #instance variables

self.age = age

myShark = Shark("Mark",13)

print(myShark.animal\_type)

import geometry

from geometry import Point

pt1 = Point(20,20)

pt = geometry.Point(10, 20)

print(pt)

pt.\_\_bytes\_\_()

pt0 = geometry.Point.getOrigin()

print(pt0)

x = pt.distance(geometry.Point(20, 10))

print(x)

Lists

listA = ["John", "Jane"];

tupleB = (listA, 1, "two" );

listA[0]="James"

print(listA) # prints ['James', 'Jane']

print(tupleB) # prints (['James', 'Jane'], 1, “two”)

tupleB[0][0]="John"

print(listA) # prints ['John', 'Jane']

print(tupleB) # prints (['John', 'Jane'], 1, “two”)

#tupleB[1]=3 # prints error, tuple is immutable

listA = ["Hello",2,"World"]

print(len(listA))

listA.remove(2)

listA.insert(1,"Wonderful")

listA.append("!")

print(listA)

print(listA.pop(len(listA) -1))

print(listA)

for x in listA:

print(x)

for x in range(0, 3):

print ("We're on iteration %d" % x)

listB = [1,2,3,4,5,6,7,8,9,10]

print(listB[slice(0,9,3)])

print(listB[0:9:3])

print(listB[3:5:])

print(listB[:5:])

v = (8,9,10)

print(v)

x,y,z = 1,2,3

w = x,y,z

print(w)

a,b,c = w

print(b)

DeepVsShallow

import copy

class City:

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

self.name=name

class University:

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

self.name=name;self.city=city

rva=City("Richmond")

vcu=University("VCU",rva)

print(id(vcu), id(vcu.name), id(vcu.city), id(vcu.city.name))

spawnU=vcu

#print(id(spawnU), id(spawnU.name), id(spawnU.city), id(spawnU.city.name))

spawnU=copy.copy(vcu)

#print(id(spawnU), id(spawnU.name), id(spawnU.city), id(spawnU.city.name))

spawnU=copy.deepcopy(vcu)

print(id(spawnU), id(spawnU.name), id(spawnU.city), id(spawnU.city.name))

spawnU.city.name = "Roanoke"

print(id(spawnU), id(spawnU.name), id(spawnU.city), id(spawnU.city.name))

print(id(vcu), id(vcu.name), id(vcu.city), id(vcu.city.name))