 **Northwestern Polytechnic University**

**Python Programming**

**Homework Assignment #7**

**Due day: 11/30/2021**

**Instruction:**

1. **Push the source code to Github or answer sheet in word file**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework submission could not be accepted.**

**4. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. Complete the *reiterate* method in the following *People*class to repeat what he/she said

Notice that the other methods in the class are needed to change as well, not just the *reiterate* method.

*class People ():*

*"""*

*>>> michael = People("Michael")*

*>>> michael.reiterate() # starts at whatever value you'd like*

*'I am reading lecture handout!'*

*>>> michael.say("Python")*

*'Python'*

*>>> michael.reiterate()*

*''Python'*

*>>> michael.greet()*

*'Hi, this is Michael'*

*>>> michael.reiterate()*

*'Hi, this is Michael'*

*>>> michael.ask('discuss about Python programming')*

*'Would you please discuss about Python programming'*

*>>> michael.reiterate()*

*'Would you please discuss about Python programming'*

*"""*

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

*self.name = name*

*def say(self, sth):*

*return sth*

*def ask(self, sth):*

*return self.say("Would you please " + sth)*

*def greet(self):*

*return self.say("Hi, this is " + self.name)*

*def reiterate(self): # Your program*

*… …*

class People():

  def \_\_init\_\_(self, name):

    self.name = name

    self.sth = "I am reading lecture handout!"

  def say(self, sth):

    self.sth = sth

    return sth

  def ask(self, sth):

    return self.say("Would you please " + sth)

  def greet(self):

    return self.say("Hi, this is " + self.name)

  def reiterate(self):

    return self.say(self.sth)

  def ask(self, sth):

    return self.say("Would you please " + sth)

michael = People("Michael")

print(michael.reiterate())

print(michael.say("Python"))

print(michael.reiterate())

print(michael.greet())

print(michael.reiterate())

print(michael.ask('discuss about Python programming'))

print(michael.reiterate())

1. Assuming that a class called *Twice* will be designed to represent people who always say something twice:

*>>> michael = Twice("Michael")*

*>>> michael.say("hi")*

*"hi hi"*

*>>> michael.say("Python programming is interesting")*

*"Python programming is interesting Python programming is interesting"*

Consider the following three classes for *Twice* that inherit from the *People* class:

*class Twice(People):*

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

*People.\_\_init\_\_(self, name)*

*def say(self, sth):*

*return People.say(self, sth) + " " + self.reiterate()*

*class Twice(People):*

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

*People.\_\_init\_\_(self, name)*

*def say(self, sth):*

*return sth + " " + sth*

*class Twice(People):*

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

*People.\_\_init\_\_(self, name)*

*def say(self, sth):*

*return People.say(self, sth + " " + sth)*

Determine which of above three classes will get what you want. Also determine for which of the methods the three versions would respond differently. (Don't forget about the *reiterate* method!)

1.

*class Twice(People):*

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

*People.\_\_init\_\_(self, name)*

*def say(self, sth):*

*return People.say(self, sth) + " " + self.reiterate()*

#The above class uses “ reiterate()”  generates a type of recursion that ends in a forever loop.

2.

*class Twice(People):*

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

*People.\_\_init\_\_(self, name)*

*def say(self, sth):*

*return sth + " " + sth*

#The second one works but the return statement doesnt follow the OOP style results in non-escalable implementation.

3.

class Twice(People):

  def \_\_init\_\_ (self, name):

    People.\_\_init\_\_(self, name)

  def say(self, sth):

    return People.say(self, sth + " " + sth)

michael = Twice("Michael")

print(michael.say("hi"))

print(michael.say("Python programming is interesting"))

#The third one works and returns the desirable output.

1. Write a class for vending machine operations in python as follows

*class VndMchn:*

*"""A vending machine operations*

*>>> a = VndMchn('soda', 3.5) # setup price $3.5 for soda*

*>>> a.vending()*

*'Out of stock currently'*

*>>> a.adding(2) # add 2 soda to machine*

*'Current soda stock: 2'*

*>>> a.vending()*

*'Need to deposit $3.5 more.'*

*>>> a.deposit(1.5) # deposit $1.5*

*'Current balance: $1.5'*

*>>> a.vending()*

*'Need to deposit $2 more.'*

*>>> a.deposit(5)*

*'Current balance: $6.5'*

*>>> a.vending()*

*'Take your soda and $3 change' # return all your change*

*>>> a.deposit(3.5)*

*'Current balance: $3.5'*

*>>> a.vending()*

*'Take your soda'*

*>>> a.deposit($10)*

*'Out of stock. Return to your $10'*

*"""*

class VndMchn():

  def \_\_init\_\_(self, item, price):

    self.item = item

    self.price = price

    self.stock = 0

    self.cash = 0

  def vending(self):

    if self.stock > 0:

      if self.cash < self.price:

        return str("Need to deposit $" + str(self.price - self.cash) + " more.")

      elif self.cash == self.price:

        self.cash = 0

        self.stock -= 1

        return "Take your soda"

      else:

        tmp = self.cash - self.price

        self.cash = 0

        self.stock -= 1

        return "Take your soda and $" + str(tmp) + " change."

    else:

      return "Out of stock currently"

  def adding(self, x):

    self.stock += x

    return "Current " + self.item + " stock: " + str(self.stock)

  def deposit(self, x):

    if self.stock > 0:

      self.cash += x

      return "Current balance : $" + str(self.cash)

    else:

      return "Out of stock. Returning to your $" + str(x)

a = VndMchn('soda', 3.5)

print(a.vending())

print(a.adding(2))

print(a.vending())

print(a.deposit(1.5))

print(a.vending())

print(a.deposit(5))

print(a.vending())

print(a.deposit(3.5))

print(a.vending())

print(a.deposit(10))

1. Generate a *Keypad* class with an arbitrary number of *Bttn* (object type) as input parameters and save these *Buttons* in a dictionary. The keys in *dict* dType will be integer that denoted the position on the keypad, and the values will be the respective Button. Complete the methods in the keypad class according to the following description

*class Keypad:*

*"""*

*>>> a1 = Bttn(0, "H")*

*>>> a2 = Bttn (1, "I")*

*>>> ky = Keypad (a1, a2)*

*>>> ky.bttns[0].key*

*'H'*

*>>> ky.prs(1) # press position 1 on the keypad*

*'I'*

*>>> ky.type([0, 1]) # typing string by pressing two keys on the keyboard*

*'HI'*

*>>> ky.type([1, 0])*

*'IH'*

*>>> a1.pressed # "H" key is pressed twice*

*2*

*>>> a2.pressed*

*3*

*"""*

*def \_\_init\_\_(self, \*args):*

*… …*

*def prs(self, info):*

*"""Reads in a position of the button pressed, and*

*returns that button's char"""*

*… …*

*def type(self, type\_ip):*

*"""Reads in a list of positions of buttons pressed, and*

*returns a string that button will generate"""*

*… …*

*class Bttn: # Button class*

*def \_\_init\_\_(self, pstn, ky):*

*self.pos = pstn*

*self.key = ky*

*self.pressed = 0*

class Keypad():

  def \_\_init\_\_(self, \*args):

    self.bttns = {}

    for ar in args:

      self.bttns[ar.pos] = ar

  def prs(self, info):

    try:

      self.bttns[info].pressed += 1

      return self.bttns[info].key

    except:

      print('Dude!, Bttn doesnt exist')

      pass

  def type(self, type\_ip):

    try:

      res = ""

      for i in type\_ip:

        self.bttns[i].pressed += 1

        res += self.bttns[i].key

      return res

    except:

      print('Bttn doesnt exist')

      pass

class Bttn(): # Button class

  def \_\_init\_\_(self, pstn, ky):

    self.pos = pstn

    self.key = ky

    self.pressed = 0

a1 = Bttn(0, 'H')

a2 = Bttn(1, 'I')

ky = Keypad(a1, a2)

print(ky.bttns[0].key)

print(ky.prs(1))

print(ky.type([0, 1]))

print(ky.type([1, 0]))

print(a1.pressed)

print(a2.pressed)

*\*notice that python visualization online tool is good software to either observe program execution process or debug your program at* [*http://pythontutor.com/visualize.html#mode=edit*](http://pythontutor.com/visualize.html#mode=edit)