Name-Surname: Chiho Li ID: 64011378

Introduction to Computers and Programming, SE Programme

Homework #12/1

10th November 2021

1.1 Define an abstract class **Char** with the following methods

Abstract class **Char**

- **Methods**
  - **draw(x,y):** draw a character using turtle at position (x,y)
  - **getWidth():** returns the width of the character drawn by method **draw(x,y)**

Now define the class **Char0** as a subclass of class **Char**

Class **Char0 (Char)**

- **Methods**
  - **draw(x,y):** draw the character 0 using turtle at position (x,y)
  - **getWidth():** returns the width of the image of character 0 drawn by **draw(x,y)**

Similarly, define the classes **Char1**, **Char2**, ..., **Char9** whose method **draw(x,y)** draws character 1, 2, ..., 9, respectively.

1.2 Define a method **drawNum(x),** where **x** is either a natural number or a string of digits 0, ...,9. **drawNum(x)** will draw **x** using turtle.

Hint: Create a dictionary whose keys are 0, ..., 9. Create a new object of class **Char0** and map key 0 to that object. Do similarly for the keys 1, ..., 9.

```
import abc
from turtle import*

class Char(abc.ABC):
    def __init__(self):
        self.width = 20
    def draw(self,x,y):
        penup()
        goto(x,y)
        pendown()
        setheading(0)
        pass
    def getWidth(self):
        return self.width

class Char0(Char):
    def __init__(self):
        super().__init__()
    def draw(self,x,y):
        super().draw(x,y)
        for i in range(2):
            fd(self.width)
            left(90)
            fd(self.width * 1.5)
            left(90)

class Char1(Char):
    def draw(self,x,y):
        super().draw(x,y)
        fd(self.width)
        back(self.width/2)
        left(90)
        fd(self.width * 1.5)
        left(135)
        fd(self.width / 1.5)

class Char2(Char):
    def draw(self, x, y):
        super().draw(x, y)
        fd(self.width)
        back(self.width)
        left(90)
        fd(self.width * 0.75)
        right(90)
        fd(self.width)
        left(90)
        fd(self.width * 0.75)
        left(90)
        fd(self.width)

class Char3(Char):
    def draw(self, x, y):
        super().draw(x, y)
        fd(self.width)
        left(90)
        fd(self.width * 0.75)
        left(90)
        fd(self.width)
        back(self.width)
        right(90)
        fd(self.width * 0.75)
        left(90)
        fd(self.width)

class Char4(Char):
    def draw(self, x, y):
        super().draw(x+self.width, y)
        left(90)
        fd(self.width * 1.5)
        back((self.width*1.5)/2)
        left(90)
        fd(self.width)
        right(90)
        fd((self.width*1.5)/2)

class Char5(Char):
    def draw(self, x, y):
        super().draw(x, y)
        fd(self.width)
        left(90)
        fd(self.width * 0.75)
        left(90)
        fd(self.width)
        right(90)
        fd(self.width * 0.75)
        right(90)
        fd(self.width)

class Char6(Char):
    def draw(self, x, y):
        super().draw(x, y)
        for i in range(2):
            fd(self.width)
            left(90)
            fd((self.width * 1.5)/2)
            left(90)
        left(90)
        fd(self.width * 1.5)
        right(90)
        fd(self.width)

class Char7(Char):
    def draw(self, x, y):
        super().draw(x, y+(self.width*1.5))
        fd(self.width)
        goto(x,y)

class Char8(Char):
    def draw(self, x, y):
        super().draw(x, y)
        for i in range(4):
            fd((self.width * 1.5)/2)
            left(90)
        goto(x,y+((self.width * 1.5)/2))
        for i in range(4):
            fd((self.width * 1.5)/2)
            left(90)

class Char9(Char):
    def draw(self, x, y):
        super().draw(x+(self.width), y)
        goto(x+(self.width),y+(self.width * 1.5)/2)
        goto(x,y+((self.width * 1.5)/2))
        for i in range(2):
            fd(self.width)
            left(90)
            fd((self.width * 1.5)/2)
            left(90)
def draw_true(input):
    speed(0)
    hideturtle()
    input = str(input)
    x_cords = 0
    holder = {"0": Char0(),"1": Char1(), "2": Char2(),
              "3": Char3(), "4": Char4(), "5": Char5(),
              "6": Char6(), "7": Char7(), "8": Char8(),
              "9": Char9()}
    for char in input:
        if char in holder:
            holder[char].draw(x_cords,0)
            x_cords += 20 * 1.25

    done()
```

```
draw_true(123456789)
```

2. Use Polymorphism in Python to write a program to calculate total cost of goods from the shopping basket of a customer shopping at a stationary shop.

Direction:

The basket is represented by a list of many different stationary goods, which could be a magazine, a book, ribbon, etc.

Apply function **getTotalCost(basket)** to calculate the total cost of the good in the basket.

Define classes for **StationaryGood**, **Magazine**, **Book**, and **Ribbon**.

Any magazine will be in full price, any book will have 10% discount from its price list, and any ribbon will cost 5 bahts per 1 metre.

Demonstration:
Suppose in the customer's basket, it contains 3 magazines "Computer World" each costs 70 Bahts, 2 books "Windows 7 for Beginners" each costs 200 Bahts, and 10 metres long blue ribbon, please run your program to calculate the total cost of the goods.

```
import abc
class StationaryGood(abc.ABC):
    def __init__(self,amount,price):
        self.amount = amount
        self.price = price
        self.cost = price * amount
    def get_cost(self):
        return self.cost

class Magazine(StationaryGood):
    def __init__(self,amount,price):
        super().__init__(amount,price)

class Book(StationaryGood):
    def discounted(self):
        return (super().get_cost()*0.9)

class Ribbon(StationaryGood):
    def __init__(self,length):
        super().__init__(length,5)

def getTotalCost(basket):
    total = 0
    for item in basket:
        total += item.get_cost()
    return total

basket = [Magazine(5,50),Book(3,150),Ribbon(5)]
print(f"Total: {getTotalCost(basket)}")
```

Name-Surname: Chiho Li ID: 64011378

Introduction to Computers and Programming, SE Programme

Homework #12/2

10th November 2021

1. To save time and space when sending an SMS or a tweet, some words or phrases are often abbreviated. Below is a list of commonly-used abbreviations.

be	b
because	cuz
see	c
the	da
okay	ok
are	r
you	u
without	w/o
why	y
see you	cu
ate	8
great	gr8
mate	m8
wait	w8
later	l8r
tomorrow	2mro
for	4
before	b4
once	1ce
and	&
Your, You're	ur
As far as I know	afaik
As soon as possible	ASAP
At the moment	atm
Be right back	brb
By the way	btw
For your Information	FYI
In my humble opinion	imho
In my opinion	imo
Laughing out loud	lol
Oh my god	omg
Rolling on the floor laughing	rofl
Talk to you later	ttyl

1.1 Write function **textese(s)** which, given a string s of message in plain English, returns a string resulted from replacing words or phrases in s using the above abbreviations. The abbreviated string should be as short as possible.

1.2 Write function **untextese(s)** which, given a string s of message employing the above abbreviations, returns a string of message in plain English.

2. Given two dictionaries **dict1** and **dict2**, suppose we define the composition of **dict1** and **dict2** to be the dictionary **dict3** such that a (key:value)-pair **k:v** is in **dict3** if and only if there exists some object **m** such that **k:m** is in **dict1** and **m:v** is in **dict2**.

Write a Python function **composite(dict1, dict2)** which returns the composite of the given dictionaries **dict1** and **dict2**.

For example:

```
>>> dict1 = {'a':'p', 'b':'r', 'c':'q', 'd':'p', 'e':'s'}
>>> dict2 = {'p':'1', 'q':'2', 'r':'3'}
>>> composite(dict1, dict2)
{'a':'1', 'b':'3', 'c':'2', 'd':'1'}
```

```
def composite(dict1,dict2):
    dict3 = {}
    for i in dict1.keys():
        for y in dict2.keys():
            if y == dict1[i]:
                dict3[i] = dict2[y]
    print(dict3)

dict1 = {'a':'p', 'b':'r', 'c':'q', 'd':'p', 'e':'s'}
dict2 = {'p':'1', 'q':'2', 'r':'3'}

composite(dict1,dict2)
```

3. Suppose we are given sets **s** and **t**. The Cartesian product of **s** and **t** is the set of all tuples **(x,y)** such that **x** is a member of **s** and **y** is a member of **t**.

More generally, suppose we are given **N** sets **s1**, ..., **sN** (where **N≥1**). The Cartesian product of **s1**, ..., **sN** is the set of all N-tuples (**x1**, ..., **xN**) such that **x1** is a member of **s1**, ..., and **xN** is a member of **sN**.

Write a Python function **product(s1, ..., sN)**, where **s1**, ..., **sN** are sets and N≥**1**, which returns the Cartesian product of **s1**, ..., **sN**.

For example:

```
>>> s1 = set([1, 2, 3])
>>> s2 = set([‘p’, ‘q’])
>>> s3 = set([‘a’, ‘b’, ‘c’])
>>> product(s1, s2)
set([ (1,’p’), (1, ‘q’), (2,’p’), (2, ‘q’), (3,’p’), (3, ‘q’) ])

>>> product(s1, s2, s3)
set([ (1,’p’,’a’), (1,’p’,’b’), (1,’p’,’c’), (1,’q’,’a’), (1,’q’,’b’), (1,’q’,’c’),
    (2,’p’,’a’), (2,’p’,’b’), (2,’p’,’c’), (2,’q’,’a’), (2,’q’,’b’), (2,’q’,’c’),
    (3,’p’,’a’), (3,’p’,’b’), (3,’p’,’c’), (3,’q’,’a’), (3,’q’,’b’), (3,’q’,’c’) ])
```

```
>>> product(s1)
set([ (1,), (2,), (3,)  ])
```