Introduction to OOP

(Object-Oriented Programming)

Penek Suksuda – GDSC Python Hunt Day 4 (Academic Year 2023)

An Overview of Today's Session

- An Amazing Donut Shop in a Python Land (Story)
- Explore OOP Concepts using Python
- Explore 4 Principles of Object-Oriented Programming
- A Brief overview of NumPy, pandas, matplotlib
- OOP Exercises!

a = 1

x = False

b = 1.5

c = “Hello World”

donut_1 = “glazed”

Variable

donut_2 = “mocha”

Variable

donut_3 = “chocolate”

Variable

donut_4 = “strawberry”

Variable

glazed = 10

Variable

mocha = 10

Variable

chocolate = 10

Variable

strawberry = 10

Variable

glazed = 10

Variable

mocha = 10

Variable

chocolate = 10

Variable

strawberry = 10

Variable

glazed = 10

Variable

glazed_has_hole = True

Variable

glazed_frosting = “sugar”

Variable

glazed_topping = “”

Variable

glazed = 10

Variable

mocha = 10

Variable

chocolate = 10

Variable

strawberry = 10

Variable

glazed

Object

mocha

Object

chocolate

Object

strawberry

Object

class Donut:

glazed

Object

mocha

Object

chocolate

Object

strawberry

Object

class Donut:

glazed

Object

mocha

Object

chocolate

Object

strawberry

Object

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

Instance Variable
(belongs to an instance/object)

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

total_amount
= 40

Class Variable
(belongs to a class)

class Donut:

glazed

Method

Method

Method

Method

Instance Method
(belongs to an instance/object)

class Donut:

glazed

Method

Method

Method

Method

Method

Class Method
(belongs to a class)

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

Define

```
class Donut:
    pass


x = str()                   # <class 'str'>
glazed = Donut()               # <class '__main__.Donut'>
glazed.amount = 10             # <class 'int'>
glazed.has_hole = True         # <class 'bool'>
glazed.frosting = “sugar”      # <class 'str'>
glazed.topping = “”            # <class 'str'>
```

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

class Donut:

glazed

amount = 10

frosting = “sugar”

has_hole = True

topping = “”

Constructor

Python Magic Methods

MAGIC

Class Instantiation	
`__init__(self, ...args)`	`ClassName()`
`__del__(self)`	`del instance`

Property Lookups	
`__getattr__(self, key)`	`instance.prop` (when `prop` not present)
`__getattribute__(self, key)`	`instance.prop` (regardless of `prop` present)
`__dir__(self)`	`dir(instance)`
`__setattr__(self, key, val)`	`instance.prop = newVal`
`__delattr__(self, key)`	`del instance.prop`
`__getitem__(self, key)`	`instance[prop]`
`__setitem__(self, key, val)`	`instance[prop] = newVal`
`__delitem__(self, key)`	`del instance[prop]`

List Iteration	
`__iter__(self)`	`[x for x in instance]`
`__contains__(self, item)`	`if x in instance`

Operator Overloads	
`__add__(self, other)`	`instance + other`
`__sub__(self, other)`	`instance - other`
`__mul__(self, other)`	`instance * other`
`__eq__(self, other)`	`instance == other`
`__ne__(self, other)`	`instance ≠ other`
`__lt__(self, other)`	`instance < other`
`__gt__(self, other)`	`instance > other`
`__le__(self, other)`	`instance ⩽ other`
`__ge__(self, other)`	`instance ⩾ other`

Type Casting	
`__bool__(self)`	`bool(instance)`
`__int__(self)`	`int(instance)`
`__str__(self)`	`str(instance)`

CoderPad

For a full Magic Methods guide:

bit.ly/PythonMagicMethods

Constructor

```
class Donut:
    def __init__(self, amt, has_hole, frt, tpp):
        self.amount = amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp

glazed = Donut(10, True, “sugar”, “”)
```

self

Specify Types

```
class Donut:
    def __init__(self, amt: int, has_hole: bool, frt: str, tpp: str):
        self.amount = amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp

glazed = Donut(10, True, “sugar”, “”)
```

Default Params

```
class Donut:
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self.amount = amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp

glazed = Donut(amt=10)
glazed.frosting = “sugar”
```

Methods

```
class Donut:
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self.amount = amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp
    def sold(self, amt=1):
        self.amount = self.amount - amt

glazed = Donut(amt=10, frt="sugar"); glazed.sold(4); glazed.sold()
print(glazed.amount)                                    # prints 5
```

Class Variable

```
class Donut:
    total_amount = 0
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self.amount = amt
        Donut.total_amount = Donut.total_amount + amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp
    def sold(self, amt=1):
        self.amount = self.amount - amt
        Donut.total_amount = Donut.total_amount - amt
glazed = Donut(amt=10, frt="sugar"); glazed.sold(4); glazed.sold()
print(Donut.total_amount); print(glazed.total_amount)       # Same
```

Class Method

```
class Donut:
    total_amount = 0
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        ...
    def sold(self, amt=1):
        ...
    @classmethod
    def restock_from_list(cls, list_stock):
        ...


Donut.restock_from_list([])
```

cls

Static Method

```
class Donut:
    total_amount = 0
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        ...
    def sold(self, amt=1):
        ...
    @staticmethod
    def hello(name):
            print("Hello, " + name)

Donut.hello("Penek Suksuda")
```

The 4 Principles of Object-Oriented Programming

- Abstraction
  - Hide complexity from users
- Polymorphism
  - The ability to perform a certain action in different ways
- Encapsulation
  - Safeguards the contents of a class like a capsule
- Inheritance
  - Improves code reusability, introduces sub-classes

Encapsulation

Safeguards the contents of a class like a capsule

Encapsulation

```
class Donut:
    total_amount = 0 # This needs to be considered
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self._amount = amt                          # Protected
        ...

    @property
    def amount(self):                               # Accessor
        return self._amount
    @amount.setter
    def amount(self, amt):                          # Mutator
        self._amount = amt
```

Encapsulation

```
class Donut:
    total_amount = 0 # This needs to be considered
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self.__amount = amt                          # Private
        ...

    @property
    def amount(self):                                # Accessor
        return self.__amount
    @amount.setter
    def amount(self, amt):                           # Mutator
        self.__amount = amt
```

Inheritance

Improves code reusability, introduces sub-classes

Inheritance

```
class Donut:
    total_amount = 0
    def __init__(self, amt=0, has_hole=True, frt="", tpp=""):
        self.amount = amt
        Donut.total_amount = Donut.total_amount + amt
        self.has_hole = has_hole
        self.frosting = frt
        self.topping = tpp
    ...
class MagicDonut(Donut):
    def __init__(self, amt=0, has_hole=True, frt="", tpp="", boom=False):
        super().__init__(amt, has_hole, frt, tpp)
        self.boom = boom
```

At the core of the NumPy package, is the **ndarray** object.

This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

NumPy

At the core of the NumPy package, is the **ndarray** object.

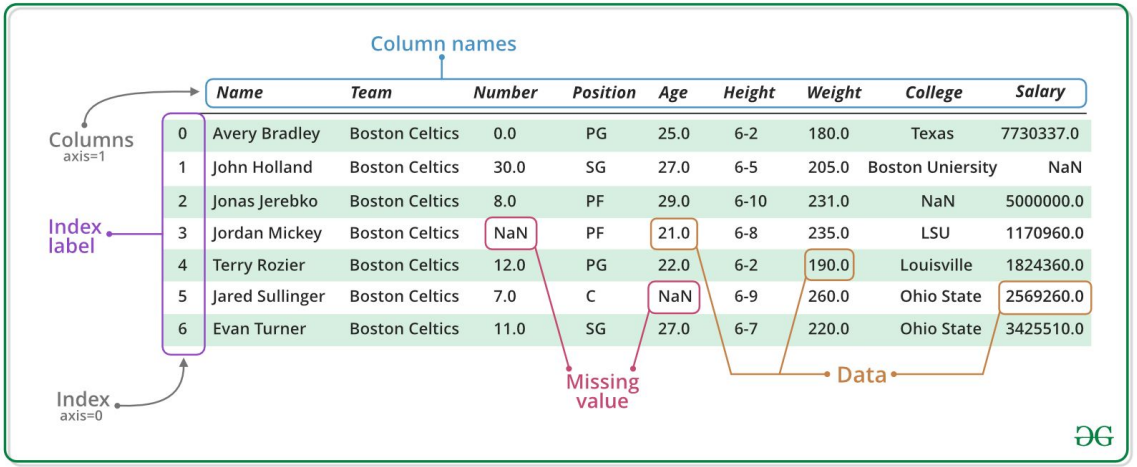
This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> np.transpose(a)
array([[1, 3],
       [2, 4]])
```

pandas

pandas is a python library for **data manipulation and analysis.**

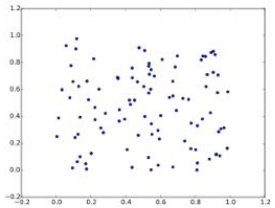
In particular, it offers **data structures and operations for manipulating numerical tables and time series.**



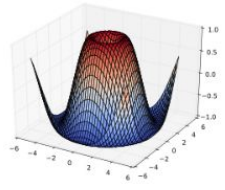
	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

matplotlib

Matplotlib is a **plotting library** for the Python programming language.



Scatter plot



3D plot

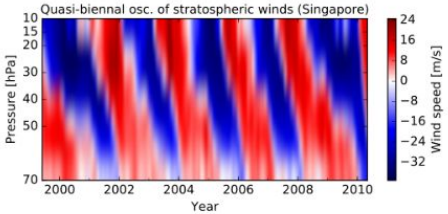
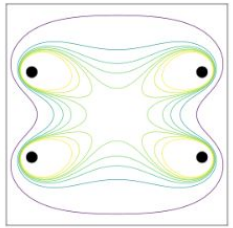
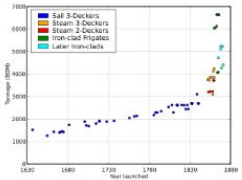


Image plot

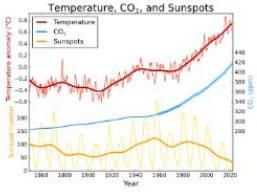


Contour plot



Scatter plot

Polar plot



Line plot

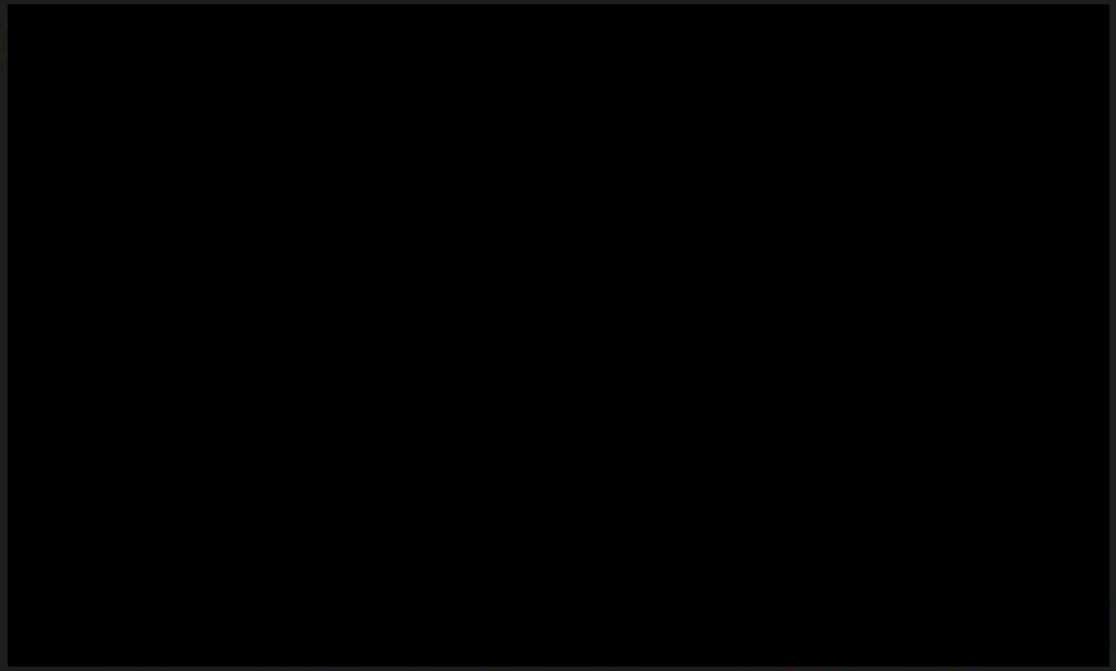
Programming Exercises

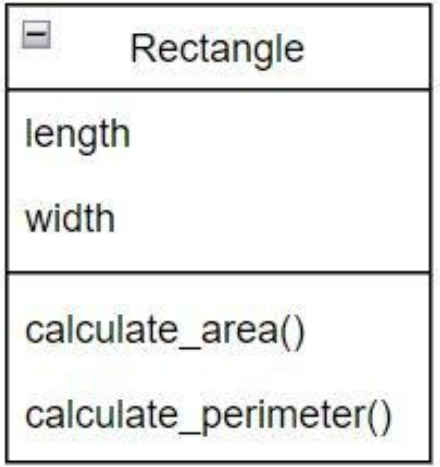
From: 2301173-Programming

Question 1 : Rectangle

สร้างคลาส Rectangle ที่สามารถคำนวณพื้นที่และเส้นรอบรูปได้

[ ]

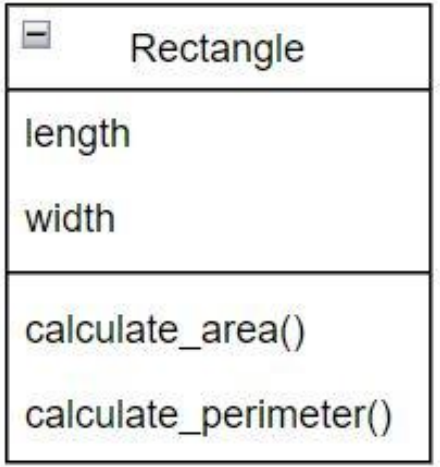




Question 1 : Rectangle

สร้างคลาส Rectangle ที่สามารถคำนวณพื้นที่และเส้นรอบรูปได้

```
[ ]  class Rectangle:
       def __init__(self, length, width):
         self.length = length
         self.width = width

       def calculate_area(self):
         return self.length * self.width

       def calculate_perimeter(self):
         return (self.length + self.width) * 2
```

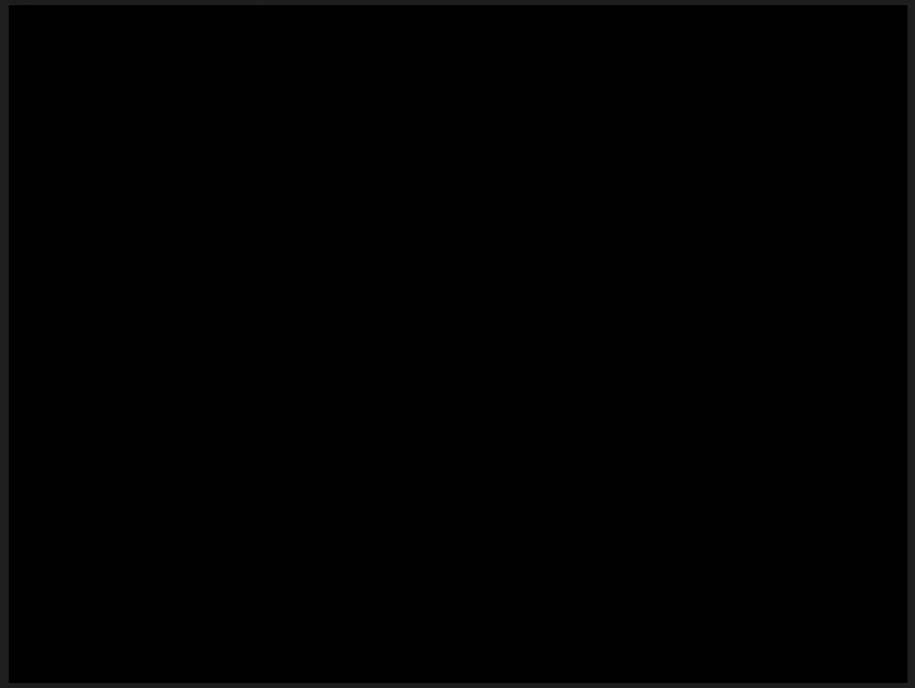


Question 2 : Shopping Cart

สร้างฟังก์ชันต่อไปนี้ของคลาส ShoppingCart ในส่วนที่กำหนดให้สมบูรณ์ เพื่อให้โปรแกรมด้านล่างทำงานได้ถูกต้องตาม test cases ต่าง ๆ

- add_item(self, item_name, qty) ฟังก์ชันสำหรับการเพิ่มรายการสินค้า (item) ลงในตะกร้า (cart) ด้วยจำนวนที่กำหนด (qty)
- remove_item(self, item_name, qty) ฟังก์ชันสำหรับการลดรายการสินค้า (item) ในตะกร้า (cart) ด้วยจำนวนที่กำหนด (qty) ถ้าสินค้าใดมีจำนวนเป็น 0 หรือติดลบ ให้เอาสินค้านั้นออกจากตะกร้าเลย
- get_total_qty(self) ฟังก์ชันสำหรับการคำนวณจำนวนรายการสินค้าทั้งหมด

[ ]



Question 2 : Shopping Cart

สร้างฟังก์ชันต่อไปนี้ของคลาส ShoppingCart ในส่วนที่กำหนดให้สมบูรณ์ เพื่อให้โปรแกรมด้านล่างทำงานได้ถูกต้องตาม test cases ต่าง ๆ

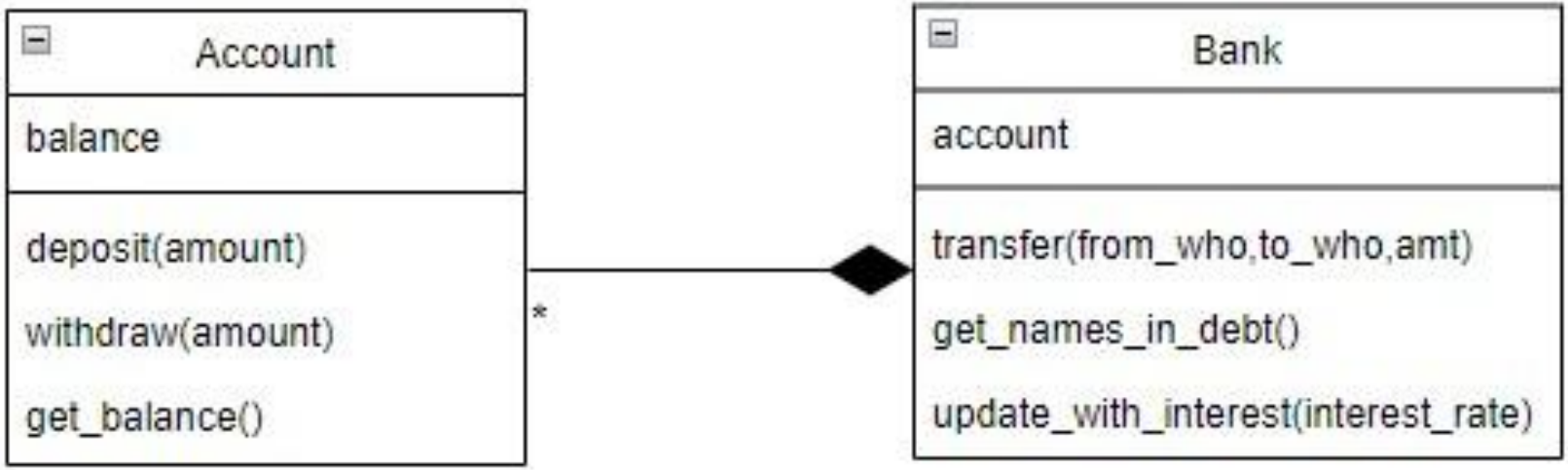
- add_item(self, item_name, qty) ฟังก์ชันสำหรับการเพิ่มรายการสินค้า (item) ลงในตะกร้า (cart) ด้วยจำนวนที่กำหนด (qty)
- remove_item(self, item_name, qty) ฟังก์ชันสำหรับการลดรายการสินค้า (item) ในตะกร้า (cart) ด้วยจำนวนที่กำหนด (qty) ถ้าสินค้าใดมีจำนวนเป็น 0 หรือติดลบ ให้เอาสินค้านั้นออกจากตะกร้าเลย
- get_total_qty(self) ฟังก์ชันสำหรับการคำนวณจำนวนรายการสินค้าทั้งหมด

```
[ ]  class ShoppingCart:
        def __init__(self):
            self.cart = {}

        def add_item(self, item, qty):
            if item in self.cart :
                self.cart.update( { item : self.cart[item] + qty } )
            else:
                self.cart.update( { item : qty } )

        def remove_item(self, item, qty):
            self.cart.update( { item : self.cart[item] - qty } )
            if self.cart[item] <= 0 :
                self.cart.pop(item)

        def get_total_qty(self):
            total = 0
            for i in self.cart.values() :
                total+=i
            return total
```

Question 3 : Bank

เขียนคลาส Bank ในส่วนที่กำหนดให้สมบูรณ์ โดยมีฟังก์ชันต่อไปนี้

- `create_accounts(self,profiles)` ฟังก์ชันสำหรับการสร้างบัญชีธนาคาร (accounts) ตาม dictionary ที่ชื่อ profiles โดย accounts มี key เป็น ชื่อบัญชี ซึ่งเป็น key ของ profiles หนึ่ง ๆ และมี value เป็นชนิด Account ที่มียอดเงินเริ่มต้น (initial_balance) เท่ากับ value ของ profiles ที่ตรงกับชื่อบัญชีนั้น ๆ เช่น
    - ถ้า profiles มีค่าเป็น {'Carol':1000,'Marta':-100} ดังนั้น accounts ที่เป็น dictionary ของคลาส Bank จะมีค่าเป็น {'Carol':Account object ที่มี initial_balance เป็น 1000,'Marta':Account object ที่มี initial_balance เป็น -100}
- `transfer(self,from_who,to_who,amt)` ฟังก์ชันสำหรับการโอนเงินจากบัญชีชื่อ from_who ไปที่บัญชีชื่อ to_who ด้วยจำนวน amt
- `get_names_in_debt(self)` ฟังก์ชันนี้คืนค่าเป็นลิสต์ของรายชื่อที่ติดหนี้กับธนาคาร นั่นคือยอดเงินในบัญชีติดลบ
- `update_with_interest(self)` ฟังก์ชันสำหรับการอัพเดททุกบัญชีในธนาคารด้วยการคิดดอกเบี้ยเงินฝากและดอกเบี้ยเงินกู้



```python
class Account:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        self.__balance -= amount

    def get_balance(self):
        return self.__balance
```

```python
class Bank:
    def __init__(self,profiles):
        self.accounts = {} # dictionary
        for i in profiles.keys() :
            self.accounts.update( { i : Account(profiles[i]) } )

    def transfer(self,from_who,to_who,amt):
        self.accounts[from_who].withdraw(amt)
        self.accounts[to_who].deposit(amt)

    def get_names_in_debt(self):
        hell = []
        for i in self.accounts.keys() :
            if self.accounts[i].get_balance() < 0 :
                hell.append(i)
        return hell

    def update_with_interest(self,interest_rate):
        for i in self.accounts.keys() :
            self.accounts[i].deposit(self.accounts[i].get_balance()*interest_rate/100)
```