香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

INTRODUCTION TO COMPUTER SCIENCE: PROGRAMMING METHODOLOGY

TUTORIAL 9

OBJECT ORIENTED PROGRAMMING(II)

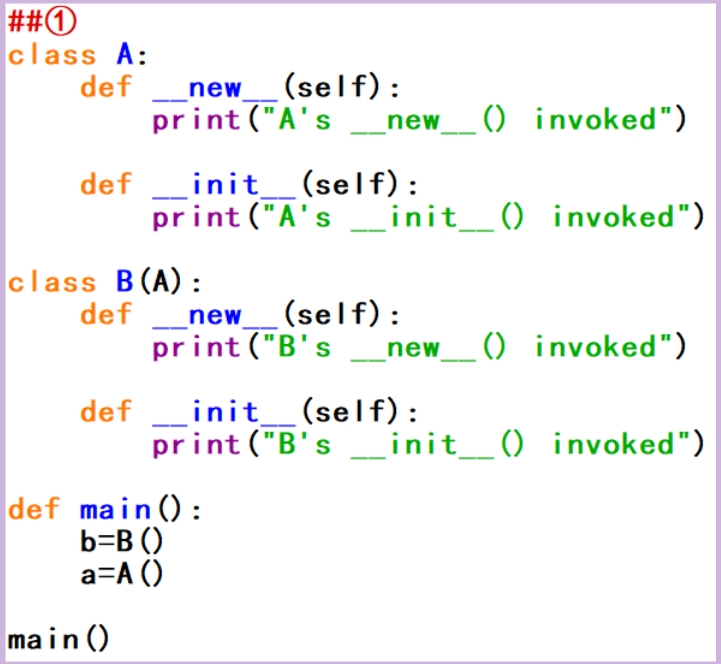
Principles of OOP

Object Oriented Programming

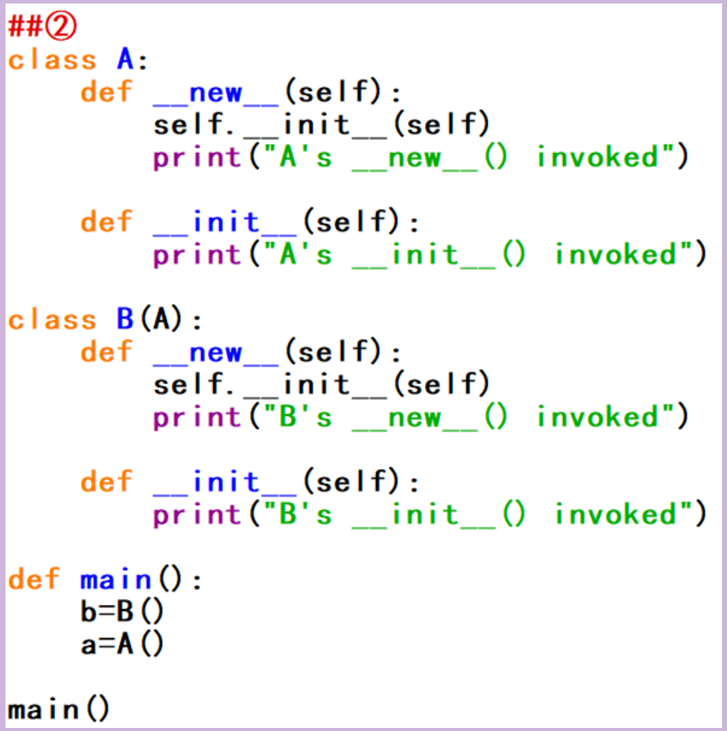
- **Abstraction(抽象):**Separate class implementation from the use of a class.
- **Encapsulation(封装):**Data inside the object should only be accessed through a public interface-the object's methods. Data and methods(functions) are encapsulated in a capsule-class. Data is usually the demanded parameters of the functions.
- **Inheritance(继承)**:Inherit the **accessible** data fields and methods from superclasses.
- **Polymorphism(多态)**:For a function using supertypes as parameters, you can pass subclass objects to it, which means the references are not determined(a result of **dynamic bounding**).
- **Dynamic Binding(动态绑定)**:A method may be implemented in several classes along the inheritance chain. The method to be invoked is dynamically bound at runtime. p.s. This is a special property for Python, other languages(e.g. Java) may not be so convenient.
- **Overriding(重载)** :The same name of methods in subclasses will replace for that in superclasses, unless it is a private method.

Q1: __new__() and __init__()

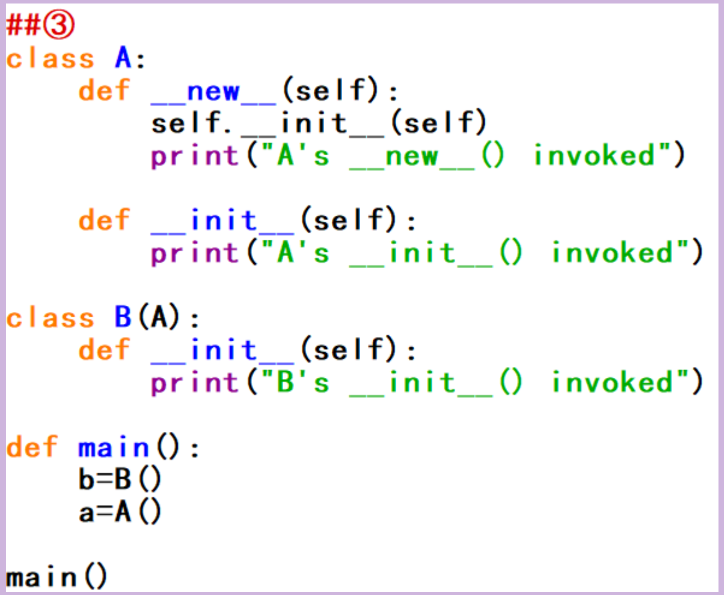
➢ What are the outputs of the following three programs respectively?



```
##①
class A:
    def __new__(self):
        print("A's __new__() invoked")

    def __init__(self):
        print("A's __init__() invoked")

class B(A):
    def __new__(self):
        print("B's __new__() invoked")

    def __init__(self):
        print("B's __init__() invoked")

def main():
    b=B()
    a=A()

main()
```



```
##②
class A:
    def __new__(self):
        self.__init__(self)
        print("A's __new__() invoked")

    def __init__(self):
        print("A's __init__() invoked")

class B(A):
    def __new__(self):
        self.__init__(self)
        print("B's __new__() invoked")

    def __init__(self):
        print("B's __init__() invoked")

def main():
    b=B()
    a=A()

main()
```

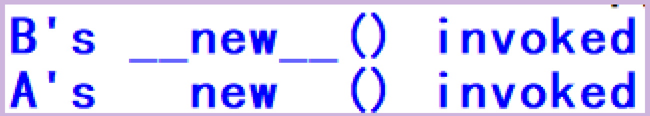


```
##③
class A:
    def __new__(self):
        self.__init__(self)
        print("A's __new__() invoked")

    def __init__(self):
        print("A's __init__() invoked")

class B(A):
    def __init__(self):
        print("B's __init__() invoked")

def main():
    b=B()
    a=A()

main()
```

➢ **Hints: i) __new__() is used to create an object, __init__() is used to initialize the data fields. ii) They are NOT private methods, which means they can be overrided.**

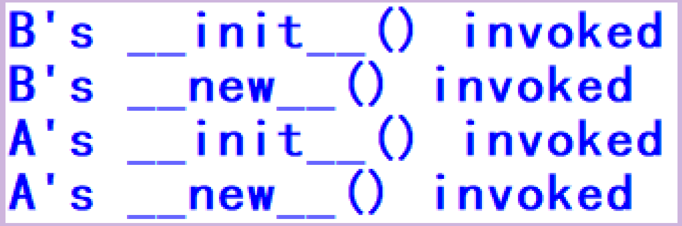
Q1: __new__() and __init__()(Answers)

- ①



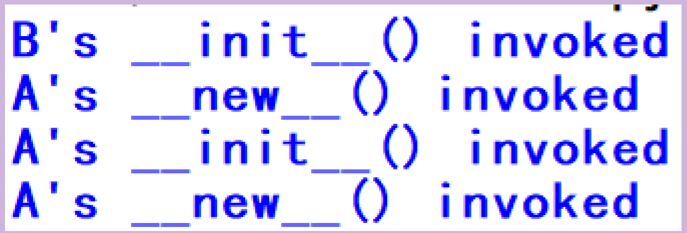
```
B's __new__() invoked
A's   new  () invoked
```

- ②



```
B's __init__() invoked
B's __new__() invoked
A's __init__() invoked
A's __new__() invoked
```

- ③

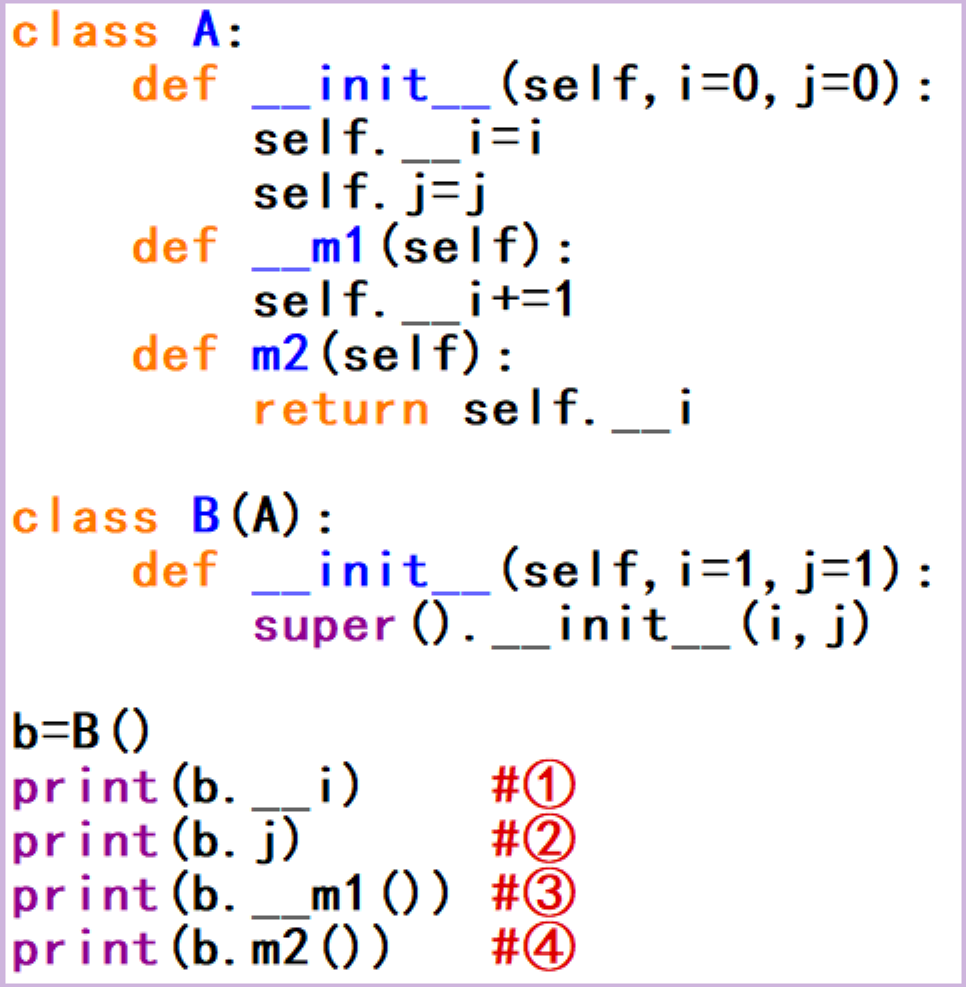


```
B's __init__() invoked
A's __new__() invoked
A's __init__() invoked
A's __new__() invoked
```

- Comments:
- i) We usually do not explicitly write __new__() method when we define a class of our own, then __new__() method in object class, which is the superclass of all classes, is invoked. And __new__() method in object class invokes __init__(), similar to that in ②③ .This is why __init__() is automatically invoked when we create an object.
- ii) Here in ① __new__() has been overrided in the way without invoking __init__(), so __init__() is not invoked in this case.
- iii) self needs to be explicitly passed to the __init__() method in __new__() method in ②③ because here "." operator means affiliation between __init__() and the class(self here means the class), instead of meaning passing parameters.

Q2: Inheritance

- According to the program on the right, which of the following statement(s) is/are correct?
  - A. 0, 0, 1, 1 will be the outputs
  - B. 1, 1, None, 2 will be the outputs
  - C. The private data field A.__i and private method A.__m1() are not inherited by B.
  - D. 1, 1 will be the outputs if line ①, ③ are deleted.



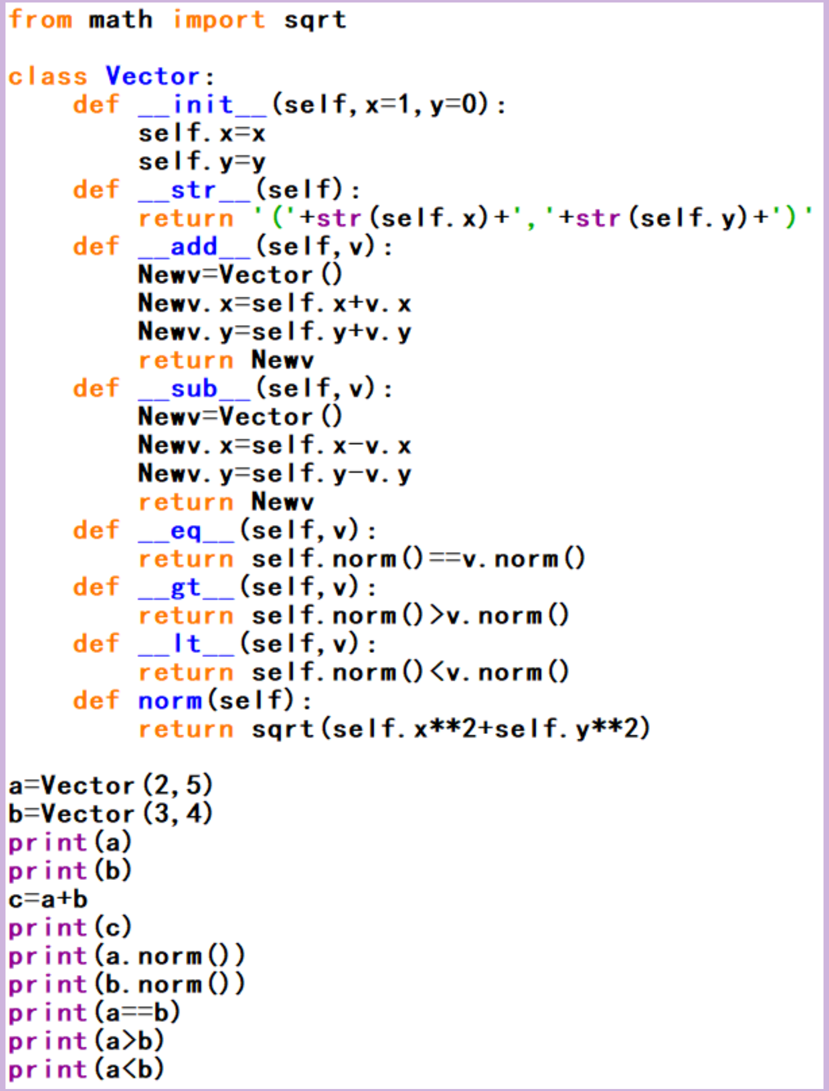
```
class A:
    def __init__(self, i=0, j=0):
        self.__i=i
        self.j=j
    def __m1(self):
        self.__i+=1
    def m2(self):
        return self.__i

class B(A):
    def __init__(self, i=1, j=1):
        super().__init__(i, j)

b=B()
print(b.__i)       #①
print(b.j)         #②
print(b.__m1())    #③
print(b.m2())      #④
```

Q2: Inheritance(Answer)

- Answer: CD
- Comments:
  - i) Only accessible data fields and methods are inherited. For private ones, they can only be used inside the class they belong to.
  - ii) super(). __init__() is needed for inheriting data fields from superclass, parameters(i and j here) can be passed to this method.
  - iii) Default values in class(B) the object(b) belong to are used, and here they are passed to super(). __init__().

Q3: Operator Symbols

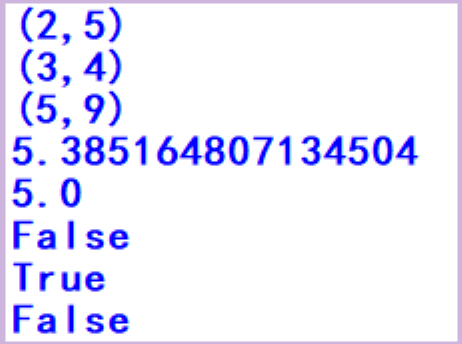
- According to the program on the right, answer the following questions.
- i) What are the outputs?
- ii) Usually when we print an object from a class we define, the result won't be the same with here. What method we define causes the difference? Is this method private?
- iii) Usually operators "+", "–","==" ,">" "<" do not support a new data type defined by ourselves, please write out the corresponding method we define for each operator. Are they private methods? What mechanism is happening here?
- iv) When doing addition and subtraction between two objects from Vector class, what data types will the results be? When doing comparison, what are being compared?



```
from math import sqrt

class Vector:
    def __init__(self,x=1,y=0):
        self.x=x
        self.y=y
    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'
    def __add__(self,v):
        Newv=Vector()
        Newv.x=self.x+v.x
        Newv.y=self.y+v.y
        return Newv
    def __sub__(self,v):
        Newv=Vector()
        Newv.x=self.x-v.x
        Newv.y=self.y-v.y
        return Newv
    def __eq__(self,v):
        return self.norm()==v.norm()
    def __gt__(self,v):
        return self.norm()>v.norm()
    def __lt__(self,v):
        return self.norm()<v.norm()
    def norm(self):
        return sqrt(self.x**2+self.y**2)

a=Vector(2,5)
b=Vector(3,4)
print(a)
print(b)
c=a+b
print(c)
print(a.norm())
print(b.norm())
print(a==b)
print(a>b)
print(a<b)
```

Q3: Operator Symbols(Answers)

- i)

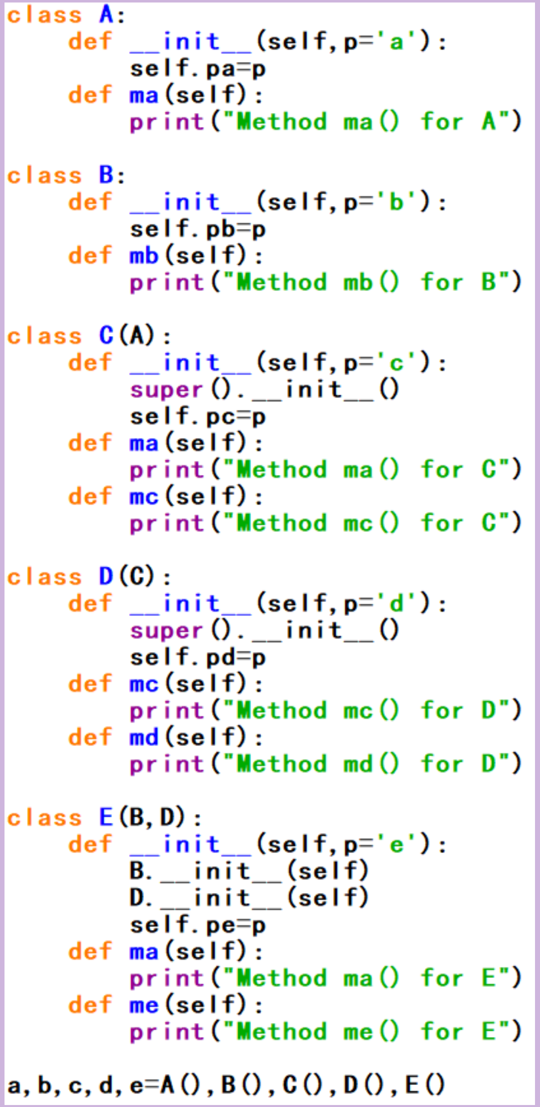


```
(2,5)
(3,4)
(5,9)
5.385164807134504
5.0
False
True
False
```

- ii)
  - __str__();
  - No.(They(“__* * *__()”) are Python’s internal methods.)
- iii)
  - “+”: __add__(), “–”: __sub__(),“==” :__eq__() ,“>” : __gt__(), “<” : __lt__() ;
  - No;
  - Overriding.
- iv)
  - A data type of Vector class which is defined here.
  - The norm of the vector.

Q4: Inheritance Tree

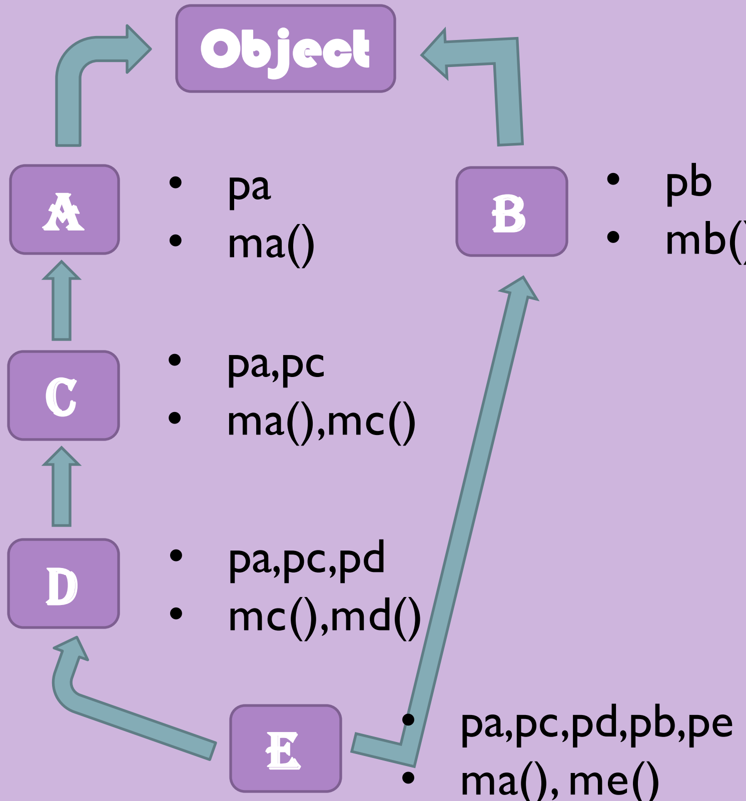
- According to the program on the right, answer the following questions.
- i) Draw the “inheritance tree” to indicate the relationship structure of the five classes defined. Layers from up toward down are from superclass to subclass.
- ii) List what data fields does each class have, and the values of them.
- iii) List what methods does each class have, and for each method, tell that which version(in which class) is invoked according to principle of Dynamic Binding.



```
class A:
    def __init__(self,p='a'):
        self.pa=p
    def ma(self):
        print("Method ma() for A")

class B:
    def __init__(self,p='b'):
        self.pb=p
    def mb(self):
        print("Method mb() for B")

class C(A):
    def __init__(self,p='c'):
        super().__init__()
        self.pc=p
    def ma(self):
        print("Method ma() for C")
    def mc(self):
        print("Method mc() for C")

class D(C):
    def __init__(self,p='d'):
        super().__init__()
        self.pd=p
    def mc(self):
        print("Method mc() for D")
    def md(self):
        print("Method md() for D")

class E(B,D):
    def __init__(self,p='e'):
        B.__init__(self)
        D.__init__(self)
        self.pe=p
    def ma(self):
        print("Method ma() for E")
    def me(self):
        print("Method me() for E")

a,b,c,d,e=A(),B(),C(),D(),E()
```

Q4: Inheritance Tree(Answers)

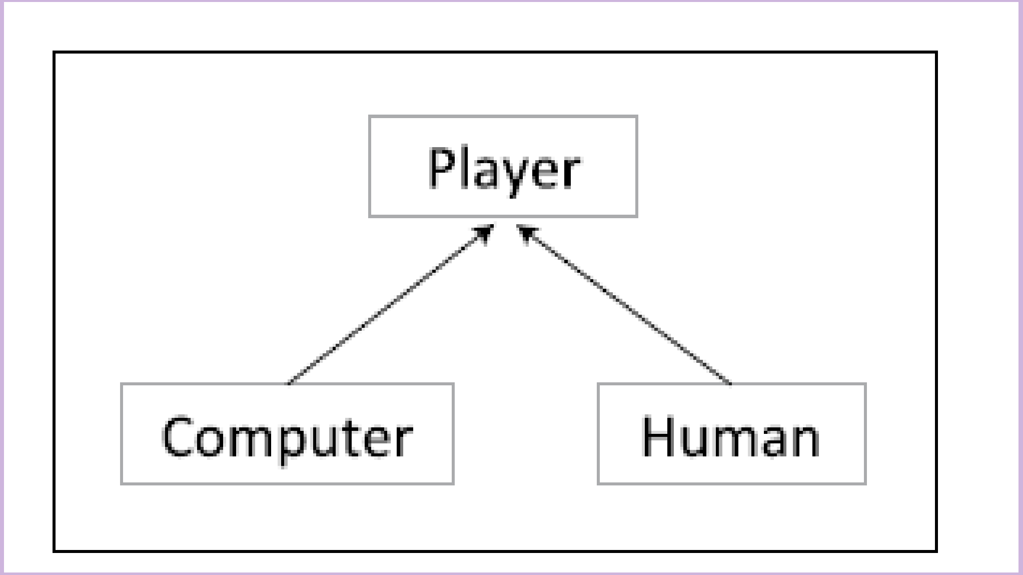
- i)



- ii)
  - a: pa='a';
  - b:pb='b';
  - c: pa='a', pc='c';
  - d: pa='a', pc='c', pd='d';
  - e:pa='a', pb='b',pc='c',pd='d',pe='e'
- iii)
  - a: A.ma();
  - b: B.mb();
  - c: C.ma(), C.mc();
  - d: C.ma(), D.mc(), D.md ();
  - e:E.ma(),B.mb(),D.mc(),D.md(),E.me()

Q5: A Simple Game: Undercut(I)

- Suppose we are creating a game that human players and computer players. Let's create a Player class that contains things common to all players, such as the score and a name.
- Let's assume that there are two kinds of players: humans and computers. The main difference is that humans enter their moves from the keyboard, whereas computers generate their moves from functions. Otherwise, they are the same, each having a name and a score.



Q5: A Simple Game: Undercut(II)

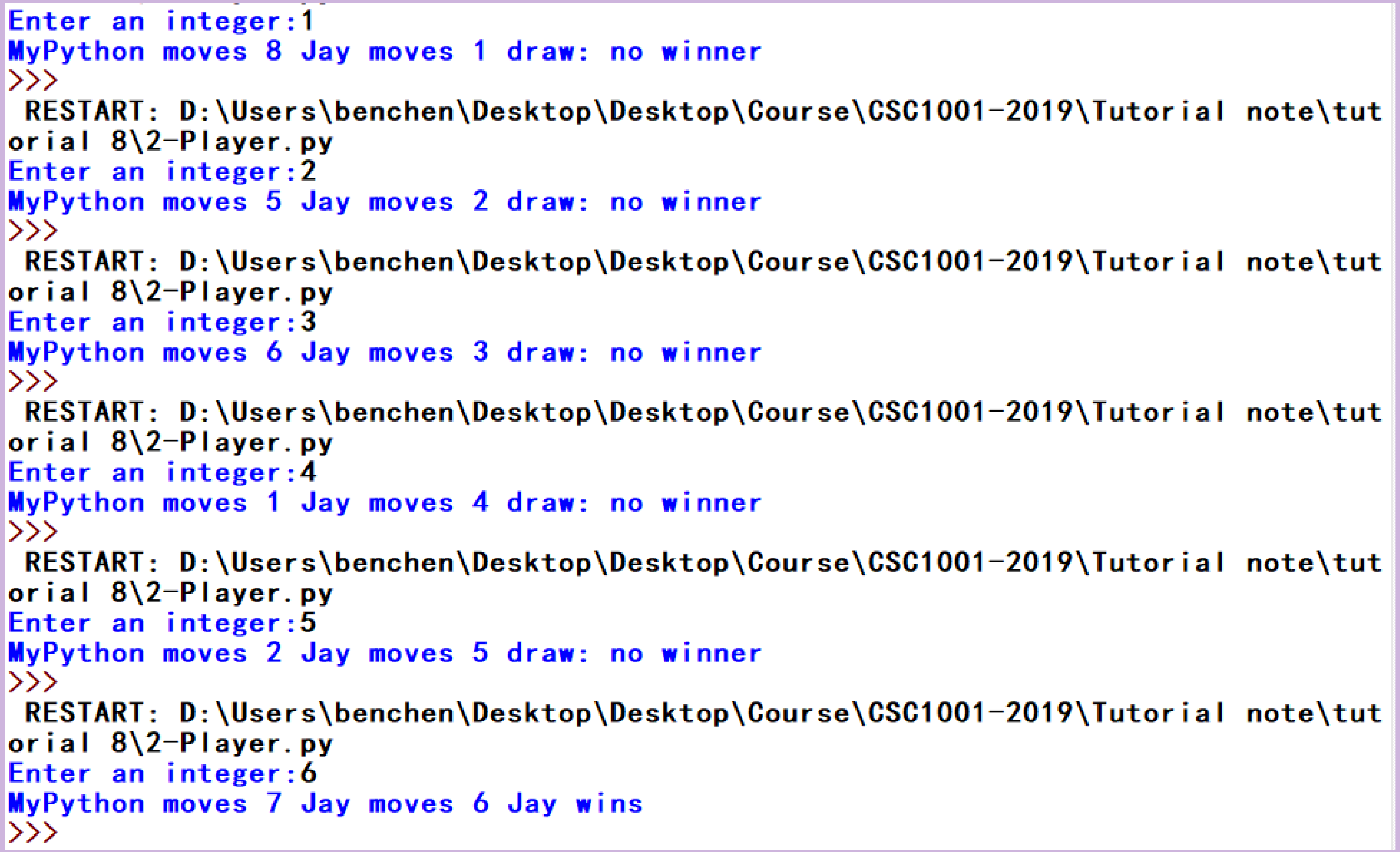
- Now, let's implement a simple game called Undercut. In undercut, two players simultaneously pick an integer from 1 to 10 (inclusive). If a player picks a number one less than the other player – if he undercuts the other by one – then he wins. Otherwise, the game is a draw. For example, if Thomas and Bonnie are playing Undercut, and they pick the numbers 9 and 10, respectively, then Thomas wins. If, instead, they choose 4 and 7, the game is a draw.
- **Hint:** Even though moves in Undercut are just numbers from 1 to 10, humans and computers determine their moves in very different ways. Human players enter a number from 1 to 10 at keyboard, whereas computer players use a function to generate their moves. Thus the human and computer classes need their own special-purpose get_move(self) methods.

Q5: A Simple Game: Undercut(III)

Step 1:
Define the getMove() method in the Human class you have defined in practice 1.
Hint: this method asks the user enter an integer from 1 to 10 and doesn't quit until the user does so.
Step 2:
Define the getMove() method in the Computer class you have defined in practice 1.
Hint: For the computer's move, we will simply have it always return a random number from 1 to 10, using random.randint(1,10) to generate a random number.
Step 3:
Define the playUnderCut() function with two parameters to indicate the two players (a human and a computer). **Polymorphism!**
Hint: In this function, firstly, two players' scores are reset to 0, and then players get their own moves. At last, the result is displayed.
Step 4:
Define the main() function, where we instance the two objects, and then call the playUnderCut() function to play this game with your computer.

Q5: A Simple Game: Undercut(IV)

- Sample runs: "MyPython" is the name for Computer player and "Jay" is that of Human player.



```
Enter an integer:1
MyPython moves 8 Jay moves 1 draw: no winner
>>>
 RESTART: D:\Users\benchen\Desktop\Desktop\Course\CSC1001-2019\Tutorial note\tut
orial 8\2-Player.py
Enter an integer:2
MyPython moves 5 Jay moves 2 draw: no winner
>>>
 RESTART: D:\Users\benchen\Desktop\Desktop\Course\CSC1001-2019\Tutorial note\tut
orial 8\2-Player.py
Enter an integer:3
MyPython moves 6 Jay moves 3 draw: no winner
>>>
 RESTART: D:\Users\benchen\Desktop\Desktop\Course\CSC1001-2019\Tutorial note\tut
orial 8\2-Player.py
Enter an integer:4
MyPython moves 1 Jay moves 4 draw: no winner
>>>
 RESTART: D:\Users\benchen\Desktop\Desktop\Course\CSC1001-2019\Tutorial note\tut
orial 8\2-Player.py
Enter an integer:5
MyPython moves 2 Jay moves 5 draw: no winner
>>>
 RESTART: D:\Users\benchen\Desktop\Desktop\Course\CSC1001-2019\Tutorial note\tut
orial 8\2-Player.py
Enter an integer:6
MyPython moves 7 Jay moves 6 Jay wins
>>>
```

Q6: Course Selection System(I)

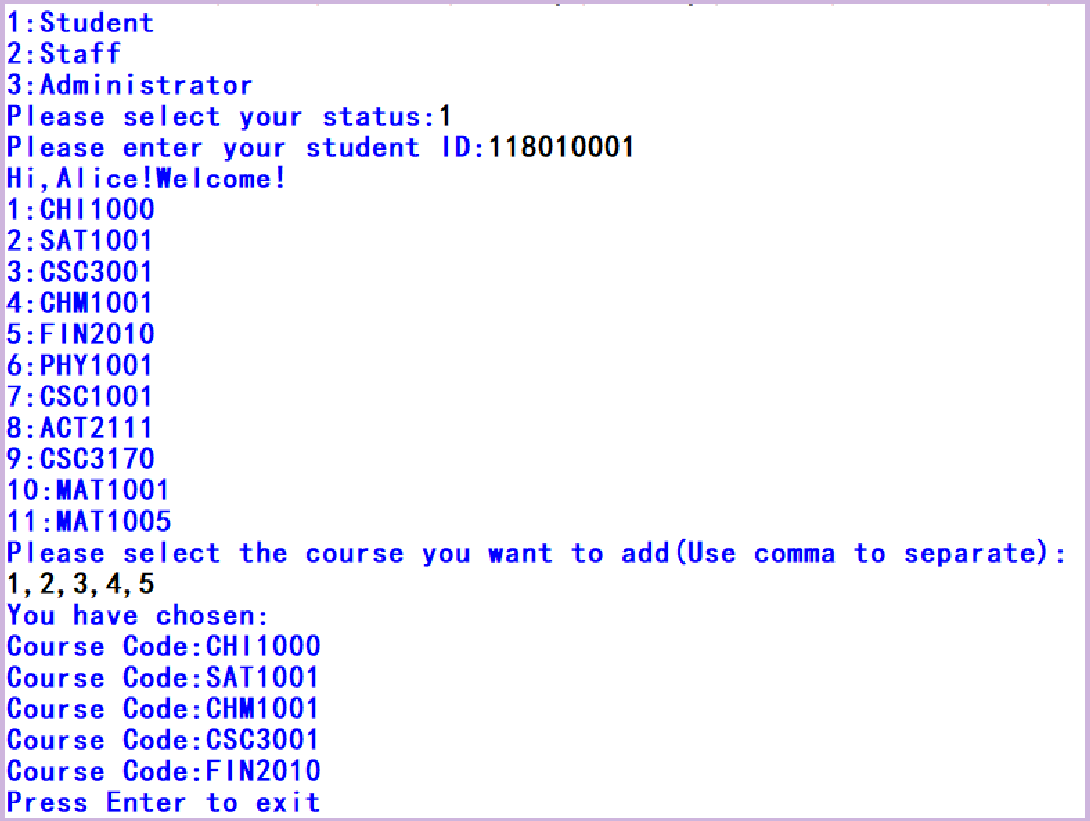
- *Association* is a general binary relationship that describes an activity between two classes. For example, a student taking a course is an association between the Student class and the Course class, and a staff teaching a course is an association between the Staff class and the Course class. Define the three classes, and in addition, a System class that has methods to simulate course selection system, import information of all students and staffs from txt files (e.g. student.txt and staff.txt here), and generate a file that summarizes the results after the selection process. You can try to write your own programs to complete this task. If no ideas, you can turn to the next page for instructions.

Q6: Course Selection System(II)

- Carefully read the readme.txt file in folder “6-Course System”, in which there are detailed description about the way I realize the simulation task, quite roughly of course. And the main data fields and methods are summarized there. If you still think it is difficult, you can use the draft codes I give to you and fulfill the blanks I leave to you, where if you put everything correctly you will be able to get the expected results. The class implementation for five classes is designed in five separately py files. Some sample runs are shown at next page.

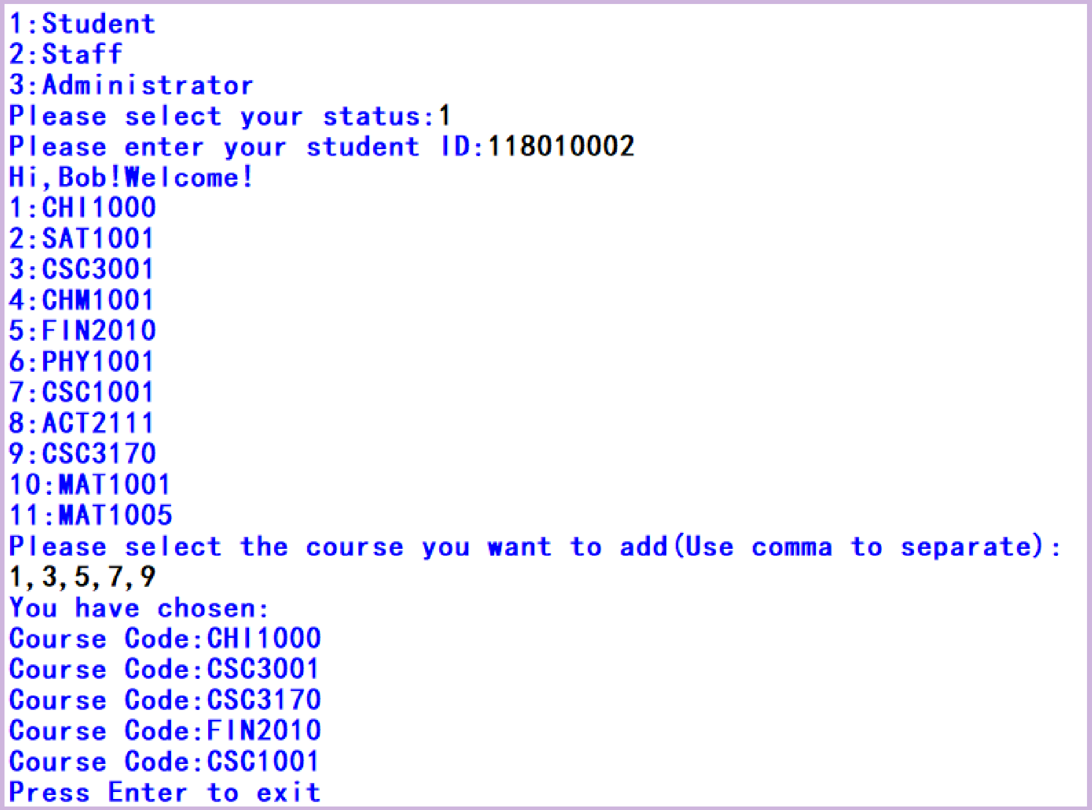
Q6: Course Selection System(III)

➢①



```
1:Student
2:Staff
3:Administrator
Please select your status:1
Please enter your student ID:118010001
Hi,Alice!Welcome!
1:CHI1000
2:SAT1001
3:CSC3001
4:CHM1001
5:FIN2010
6:PHY1001
7:CSC1001
8:ACT2111
9:CSC3170
10:MAT1001
11:MAT1005
Please select the course you want to add(Use comma to separate):
1,2,3,4,5
You have chosen:
Course Code:CHI1000
Course Code:SAT1001
Course Code:CHM1001
Course Code:CSC3001
Course Code:FIN2010
Press Enter to exit
```

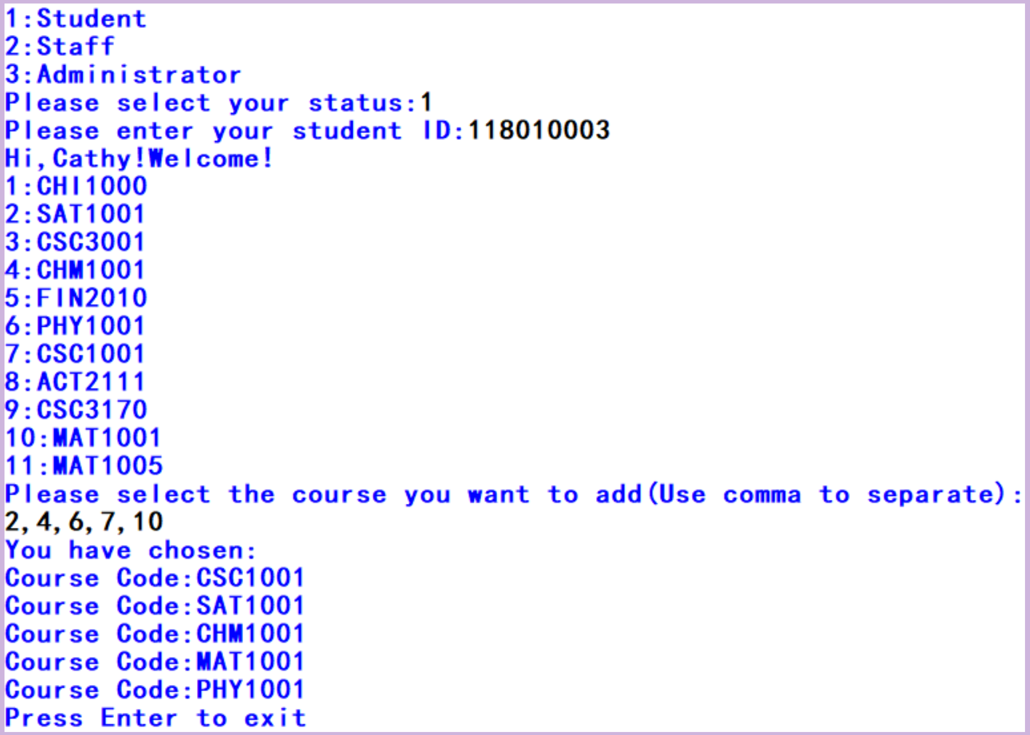
➢②



```
1:Student
2:Staff
3:Administrator
Please select your status:1
Please enter your student ID:118010002
Hi,Bob!Welcome!
1:CHI1000
2:SAT1001
3:CSC3001
4:CHM1001
5:FIN2010
6:PHY1001
7:CSC1001
8:ACT2111
9:CSC3170
10:MAT1001
11:MAT1005
Please select the course you want to add(Use comma to separate):
1,3,5,7,9
You have chosen:
Course Code:CHI1000
Course Code:CSC3001
Course Code:CSC3170
Course Code:FIN2010
Course Code:CSC1001
Press Enter to exit
```

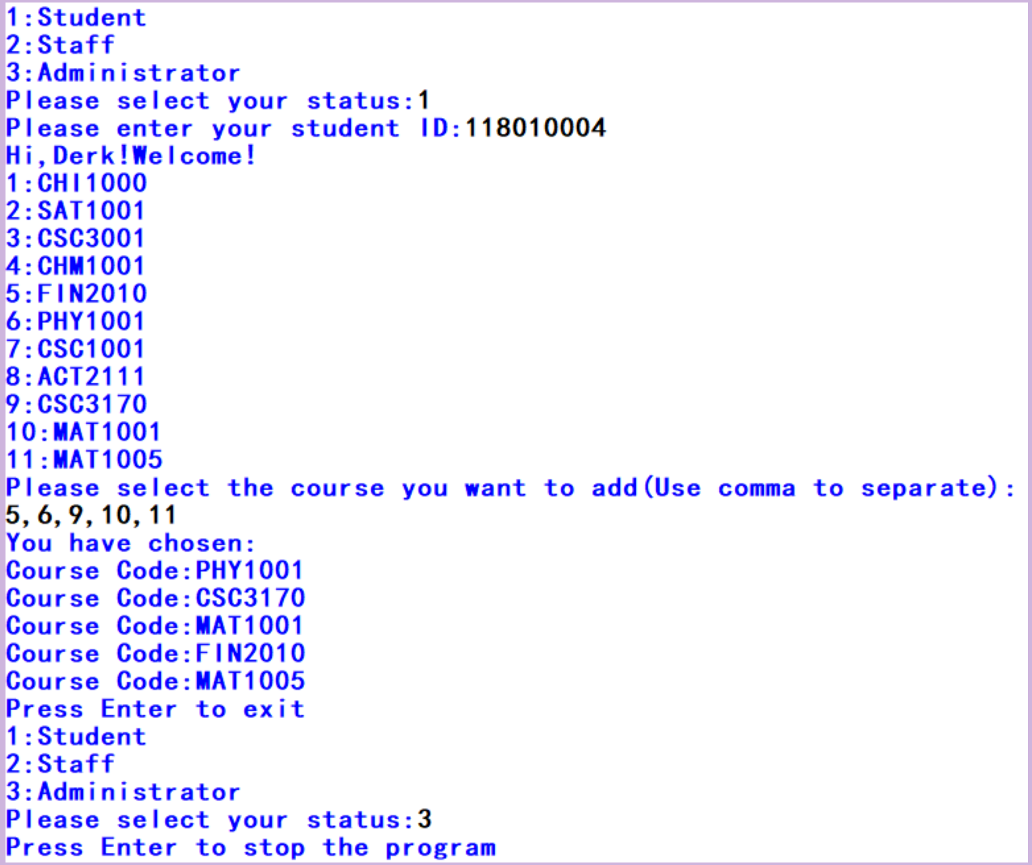
Q6: Course Selection System(IV)

➢③



```
1:Student
2:Staff
3:Administrator
Please select your status:1
Please enter your student ID:118010003
Hi,Cathy!Welcome!
1:CHI1000
2:SAT1001
3:CSC3001
4:CHM1001
5:FIN2010
6:PHY1001
7:CSC1001
8:ACT2111
9:CSC3170
10:MAT1001
11:MAT1005
Please select the course you want to add(Use comma to separate):
2,4,6,7,10
You have chosen:
Course Code:CSC1001
Course Code:SAT1001
Course Code:CHM1001
Course Code:MAT1001
Course Code:PHY1001
Press Enter to exit
```

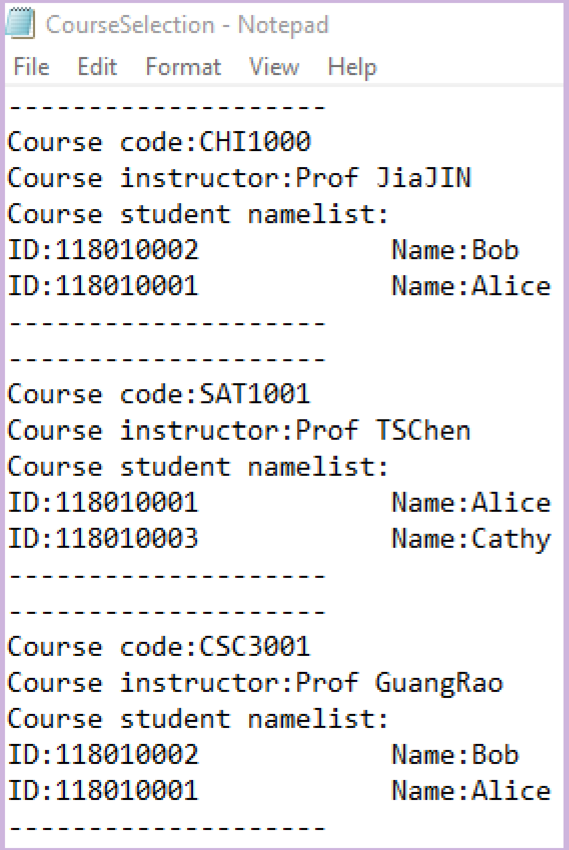
➢④



```
1:Student
2:Staff
3:Administrator
Please select your status:1
Please enter your student ID:118010004
Hi,Derk!Welcome!
1:CHI1000
2:SAT1001
3:CSC3001
4:CHM1001
5:FIN2010
6:PHY1001
7:CSC1001
8:ACT2111
9:CSC3170
10:MAT1001
11:MAT1005
Please select the course you want to add(Use comma to separate):
5,6,9,10,11
You have chosen:
Course Code:PHY1001
Course Code:CSC3170
Course Code:MAT1001
Course Code:FIN2010
Course Code:MAT1005
Press Enter to exit
1:Student
2:Staff
3:Administrator
Please select your status:3
Press Enter to stop the program
```

Q6: Course Selection System(V)

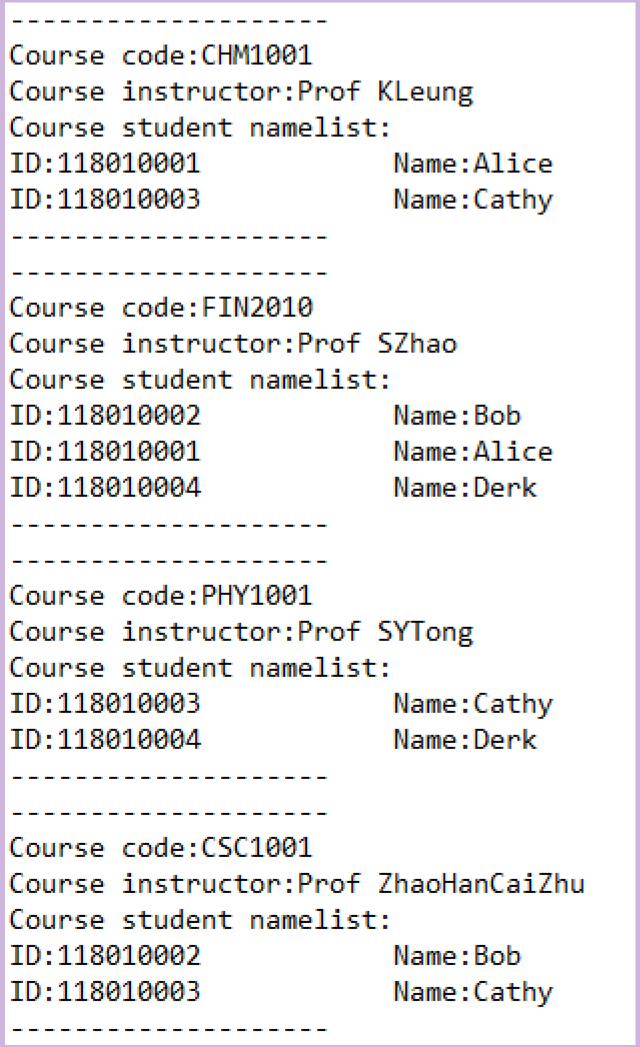
- **The CourseSelection.txt generated after course selection process:**



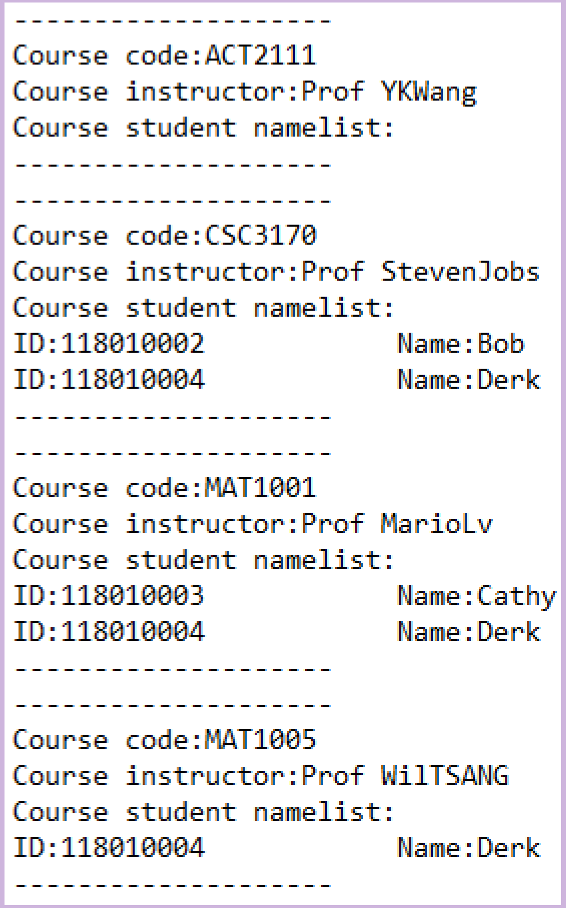
CourseSelection - Notepad

File Edit Format View Help

```
--------------------
Course code:CHI1000
Course instructor:Prof JiaJIN
Course student namelist:
ID:118010002            Name:Bob
ID:118010001            Name:Alice
--------------------
--------------------
Course code:SAT1001
Course instructor:Prof TSChen
Course student namelist:
ID:118010001            Name:Alice
ID:118010003            Name:Cathy
--------------------
--------------------
Course code:CSC3001
Course instructor:Prof GuangRao
Course student namelist:
ID:118010002            Name:Bob
ID:118010001            Name:Alice
--------------------
```



```
--------------------
Course code:CHM1001
Course instructor:Prof KLeung
Course student namelist:
ID:118010001            Name:Alice
ID:118010003            Name:Cathy
--------------------
--------------------
Course code:FIN2010
Course instructor:Prof SZhao
Course student namelist:
ID:118010002            Name:Bob
ID:118010001            Name:Alice
ID:118010004            Name:Derk
--------------------
--------------------
Course code:PHY1001
Course instructor:Prof SYTong
Course student namelist:
ID:118010003            Name:Cathy
ID:118010004            Name:Derk
--------------------
--------------------
Course code:CSC1001
Course instructor:Prof ZhaoHanCaiZhu
Course student namelist:
ID:118010002            Name:Bob
ID:118010003            Name:Cathy
--------------------
```



```
--------------------
Course code:ACT2111
Course instructor:Prof YKWang
Course student namelist:
--------------------
--------------------
Course code:CSC3170
Course instructor:Prof StevenJobs
Course student namelist:
ID:118010002            Name:Bob
ID:118010004            Name:Derk
--------------------
--------------------
Course code:MAT1001
Course instructor:Prof MarioLv
Course student namelist:
ID:118010003            Name:Cathy
ID:118010004            Name:Derk
--------------------
--------------------
Course code:MAT1005
Course instructor:Prof WilTSANG
Course student namelist:
ID:118010004            Name:Derk
--------------------
```

- **HAVE FUN!**