Python基础（二）

Python面向对象程序设计（OOP）

Python OOP

- 用户定义类型
- 方法
- 特殊方法
- 继承
- 例子：组合逻辑电路

用户定义类型

- 例子：分数类型
- C语言中的结构体

```
10
11  struct Fraction
12  {
13      int num;
14      int den;
15  };
16
17  int main(void)
18  {
19      struct Fraction frac;
20      frac.num = 3;
21      frac.den = 4;
22
23      printf("%d / %d\n", frac.num, frac.den);
24  }
25
```

用户定义类型

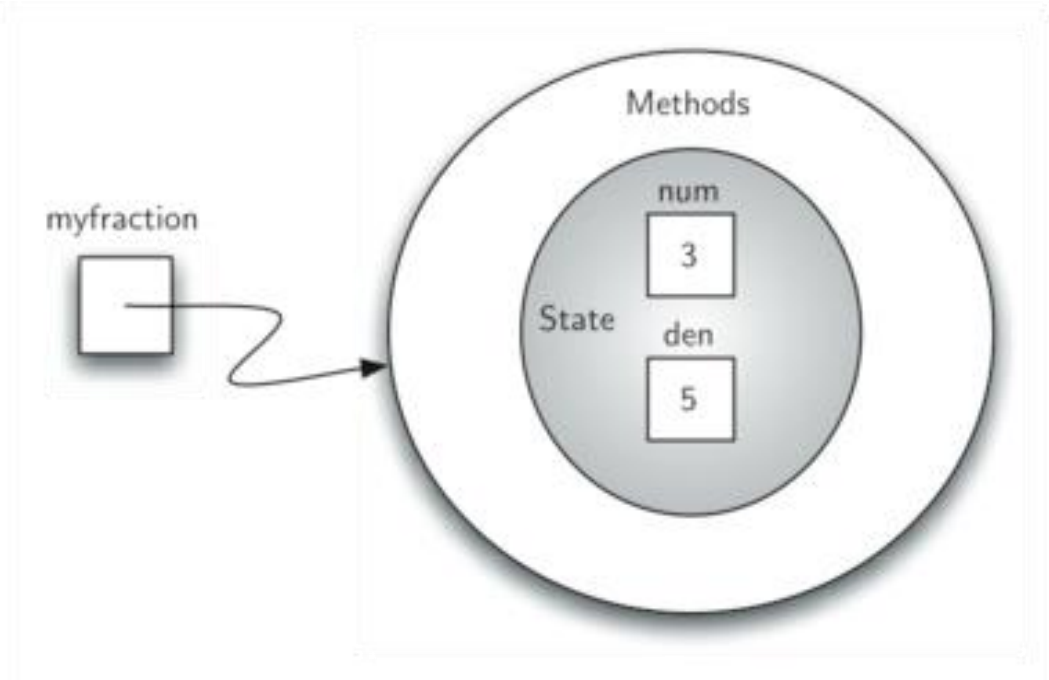
- 来到Python——类
  - 在Python中，我们通过定义一个新的类(Class)来表示一个自定义的类型

```
1
2  class Fraction:
3      pass
4
5  frac = Fraction()
6  frac.num = 3
7  frac.den = 4
8
9  print("%d / %d" % (frac.num, frac.den))
10
```

- frac是Fraction类的一个实例(Instance)

注：在Python 2中”class Fraction:”创建的是传统类型(old-style)，已不建议使用，建议使用”class Fraction(object):”；Python 3中两种写法等价创建new-style class。

用户定义类型



方法

- 在面向对象程序设计中，方法(Methods)指的是类或者是对象的一种子过程（函数）
- 列表的方法

```
>>> data = [1, 2, 3]
>>> data.append('foo')
>>> print data
[1, 2, 3, 'foo']
>>> data.pop(0)
1
>>> data
[2, 3, 'foo']
```

方法

- Python中如何实现方法？
  - 例子：输出分数（show）

```
1
2  class Fraction:
3      def show(self):
4          print("%d / %d" % (self.num, self.den))
5
6  frac = Fraction()
7  frac.num, frac.den = 3, 4
8  frac.show()
9
```

初始化方法

- `frac = Fraction()` ?
  - 把类名`Fraction`当作函数调用，将初始化并返回类的一个新实例
  - 实际的初始化方法是`__init__(self)`
- 接受参数的初始化方法

```
1
2  class Fraction:
3      def __init__(self, num, den):
4          self.num = num
5          self.den = den
6
7  frac = Fraction(3, 4)
8
```

特殊方法

- 类似__init__的特殊方法(special methods/magic methods)被用于实现特殊的语法
- 再回到输出分数：

```
>>> frac.show()
3 / 4
>>> frac
<__main__.Fraction instance at 0x0000000002CF82C8>
>>> print frac
<__main__.Fraction instance at 0x0000000002CF82C8>
```

特殊方法

- 解决方法：定义__str__

```
1
2 class Fraction:
3     def __str__(self):
4         return str(self.num) + "/" + str(self.den)
5
```

```
>>> print frac
3/4
>>> frac
<__main__.Fraction instance at 0x0000000002CF82C8>
```

  - 如何在解释环境输出frac？（__repr__方法）

特殊方法

- 对分数进行运算？

```
>>> a = Fraction(1, 4)
>>> b = Fraction(1, 2)
>>> a + b

Traceback (most recent call last):
  File "<pyshell#73>", line 1, in <module>
    a + b
TypeError: unsupported operand type(s) for +:
'instance' and 'instance'
```

特殊方法

- `a + b => a.__add__(b)`

```
1
2  class Fraction:
3      def __add__(self, other):
4          return Fraction(
5              self.num * other.den + self.den * other.num,
6              self.den * other.den
7              )
8
```

```
>>> a = Fraction(1, 4)
>>> b = Fraction(1, 2)
>>> print a + b
6/8
```

特殊方法

- 其它运算符

运算符	方法
+	__add__
-	__sub__
*	__mul__
/	__div__(__truediv__)
//	__floordiv__
%	__mod__
**	__pow__
&	__and__
|	__or__
^	__xor__
...	...

特殊方法

- 其它特殊方法
  - 基本方法：__new__、__del__、__bytes__、__format__……
  - 比较方法：__lt__、__le__、__eq__、__ne__……
  - 其它：__getitem__、__setitem__、__call__……
  - 参考资料
    - Python2 - https://docs.python.org/2/reference/datamodel.html#special-method-names
    - Python3 - https://docs.python.org/3/reference/datamodel.html#specialnames

特殊方法

- `a == b => a.__eq__(b)`

```
1
2  class Fraction:
3      def __eq__(self, other):
4          return self.num * other.den == self.den * other.num
5
```

```
>>> a = Fraction(2, 8)
>>> b = Fraction(1, 4)
>>> a == b
True
```

Fraction类，What’s next?

- 其它运算？
  - a - b, a * b, a / b, -a ...
- 约分？

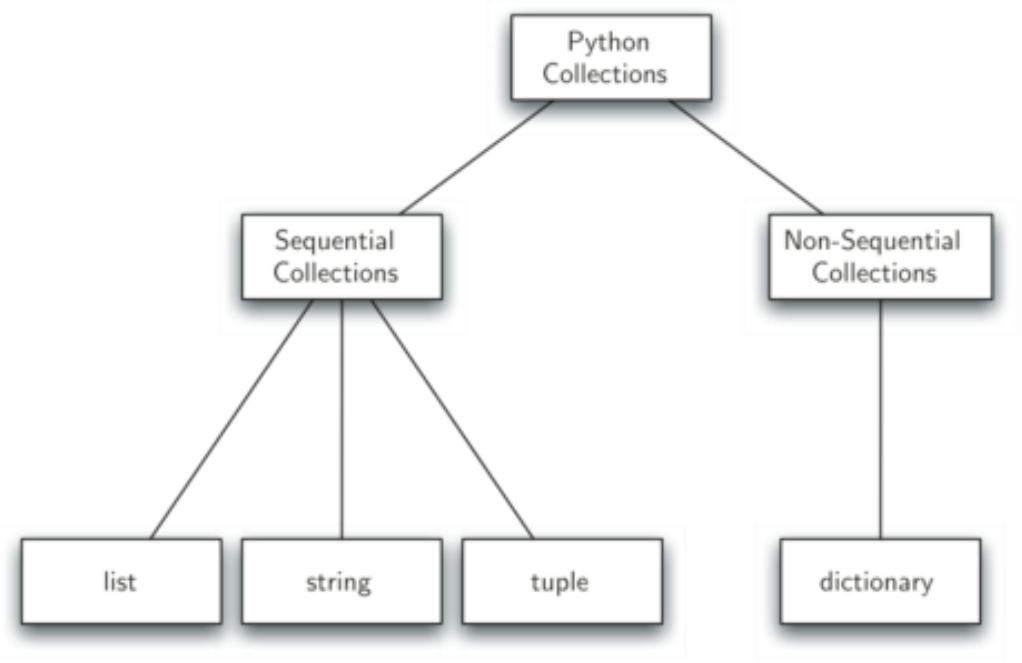
```
>>> a = Fraction(8, 12)
>>> print a
2/3
```

继承

- 继承（Inheritance）是面向对象软件技术当中的一个概念。继承是指一个类型D基于另一个类型B，具有与类型B相同的实现或行为
- 如果一个类A“继承自”另一个类B，就把这个A称为“B的子类”，而把B称为“A的父类”，也可以称“B是A的超类”。继承可以使得子类具有父类的各种属性和方法，而不需要再次编写相同的代码

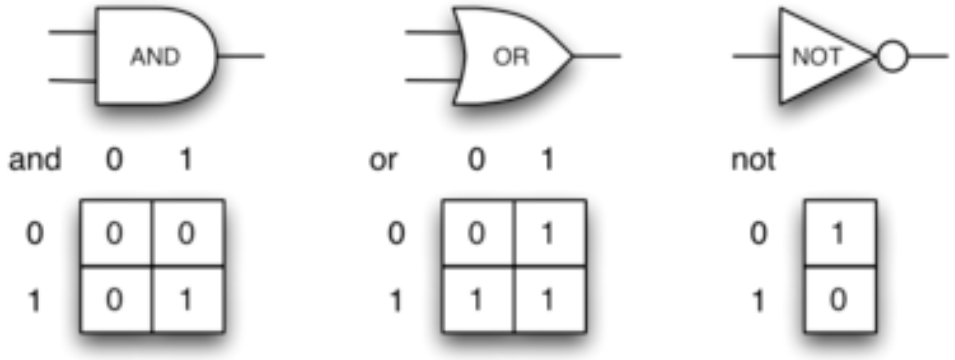
继承

- Python内建集合类型



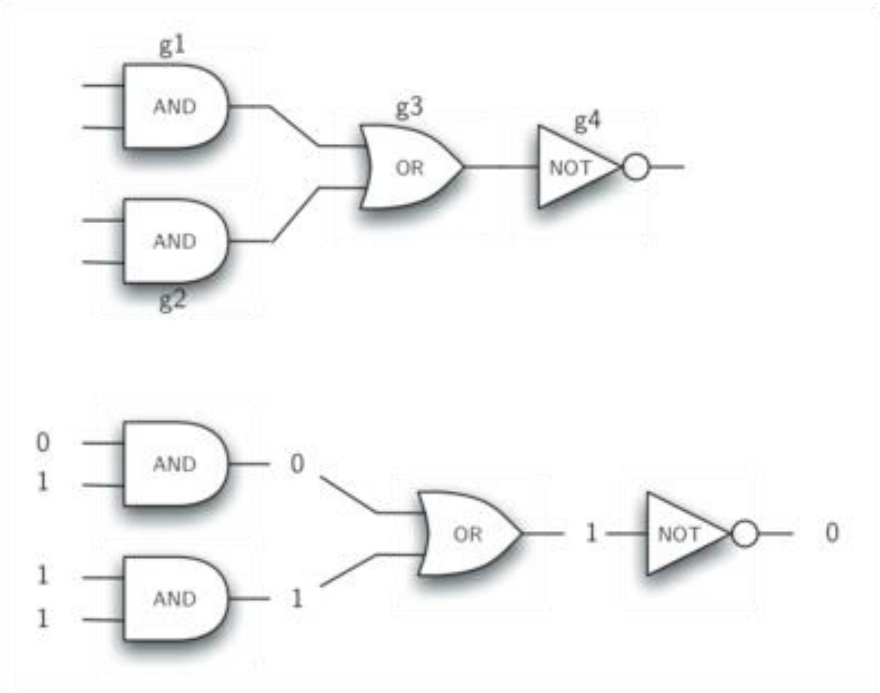
例子：组合逻辑电路

- 组合逻辑电路是由逻辑门连接组成，有固定数目的输入和输出，执行某种函数功能。输入输出为布尔值，取自集合{0, 1}，其中0表示FALSE（假），1表示TRUE（真）。
- 与门、或门、非门及其真值表



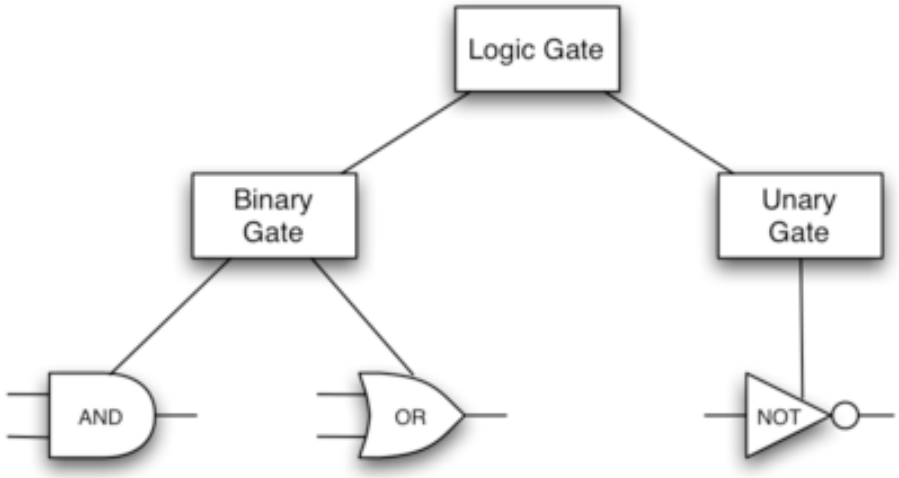
例子：组合逻辑电路

- 一个实际的组合逻辑电路
  - 四个输入，一个输出



例子：组合逻辑电路

- 逻辑门的抽象
  - 二元逻辑电路/一元逻辑电路



例子：组合逻辑电路

- 逻辑门——LogicGate类
  - 状态：label, output
  - 方法：getLabel, getOutput, performGateLogic

```
1
2  class LogicGate(object):
3      def __init__(self,n):
4          self.label = n
5          self.output = None
6
7      def getLabel(self):
8          return self.label
9
10     def getOutput(self):
11         self.output = self.performGateLogic()
12         return self.output
13
```

例子：组合逻辑电路

- 一元逻辑门——UnaryGate类
  - 状态：pin
  - 方法：getPin

```
13
14  class UnaryGate(LogicGate):
15
16      def __init__(self, n):
17          LogicGate.__init__(self, n)
18
19          self.pin = None
20
21      def getPin(self):
22          return int(input("Enter Pin input for gate "
23              + self.getLabel() + "-->"))
24
```

例子：组合逻辑电路

- 二元逻辑门——BinaryGate类
  - 状态：pinA, pinB
  - 方法：getPinA, getPinB

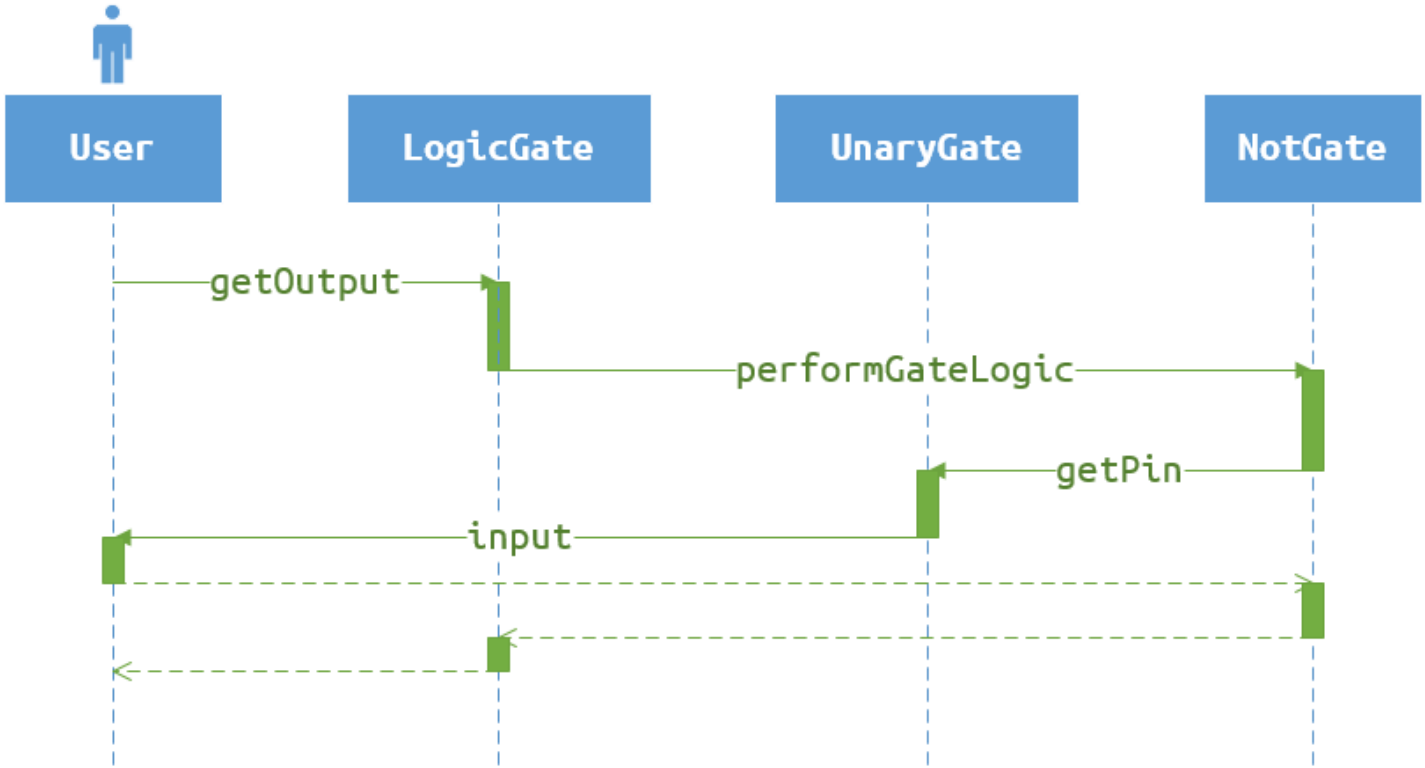
```
24
25 class BinaryGate(LogicGate):
26
27     def __init__(self, n):
28         LogicGate.__init__(self, n)
29
30         self.pinA = None
31         self.pinB = None
32
33     def getPinA(self):
34         return int(input("Enter Pin A input for gate "
35             + self.getLabel()+"-->"))
36
37     def getPinB(self):
38         return int(input("Enter Pin B input for gate "
39             + self.getLabel()+"-->"))
40
```

例子：组合逻辑电路

- 非门——NotGate
  - 实现performGateLogic

```
40
41  class NotGate(UnaryGate):
42
43      def __init__(self, n):
44          UnaryGate.__init__(self, n)
45
46      def performGateLogic(self):
47          p = self.getPin()
48          if p == 1:
49              return 0
50          else:
51              return 1
52
```

例子：组合逻辑电路


User
LogicGate
UnaryGate
NotGate
getOutput
performGateLogic
getPin
input

例子：组合逻辑电路

- 与门

```
52
53  class AndGate(BinaryGate):
54
55      def __init__(self, n):
56          BinaryGate.__init__(self, n)
57
58      def performGateLogic(self):
59          a = self.getPinA()
60          b = self.getPinB()
61          if a == 1 and b == 1:
62              return 1
63          else:
64              return 0
65
```

- 或门？

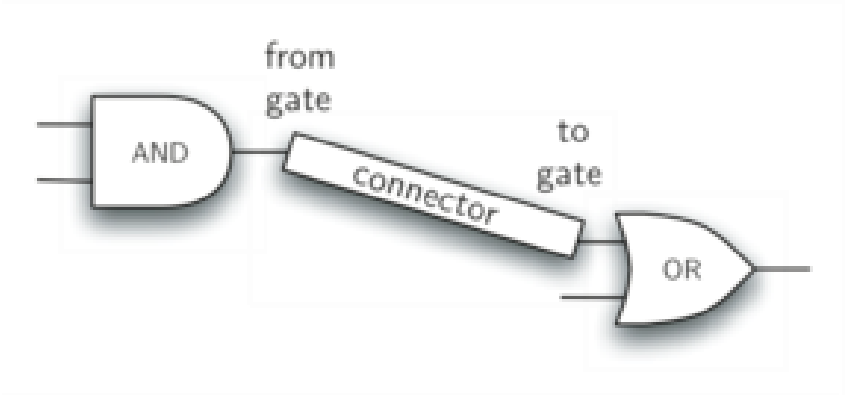
例子：组合逻辑电路

- 创建并使用与门的实例(instance)
  - 与门的状态：label, output; pinA, pinB
  - 与门的方法：getOutput, getLabel; getPinA, getPinB

```
>>> g1 = AndGate('G1')
>>> g1.getOutput()
Enter Pin A input for gate G1-->0
Enter Pin B input for gate G1-->1
0
```

例子：组合逻辑电路

- 将逻辑门组合起来—连接器Connector类



例子：组合逻辑电路

- 思路：让to-gate实现连接（setNextPin）

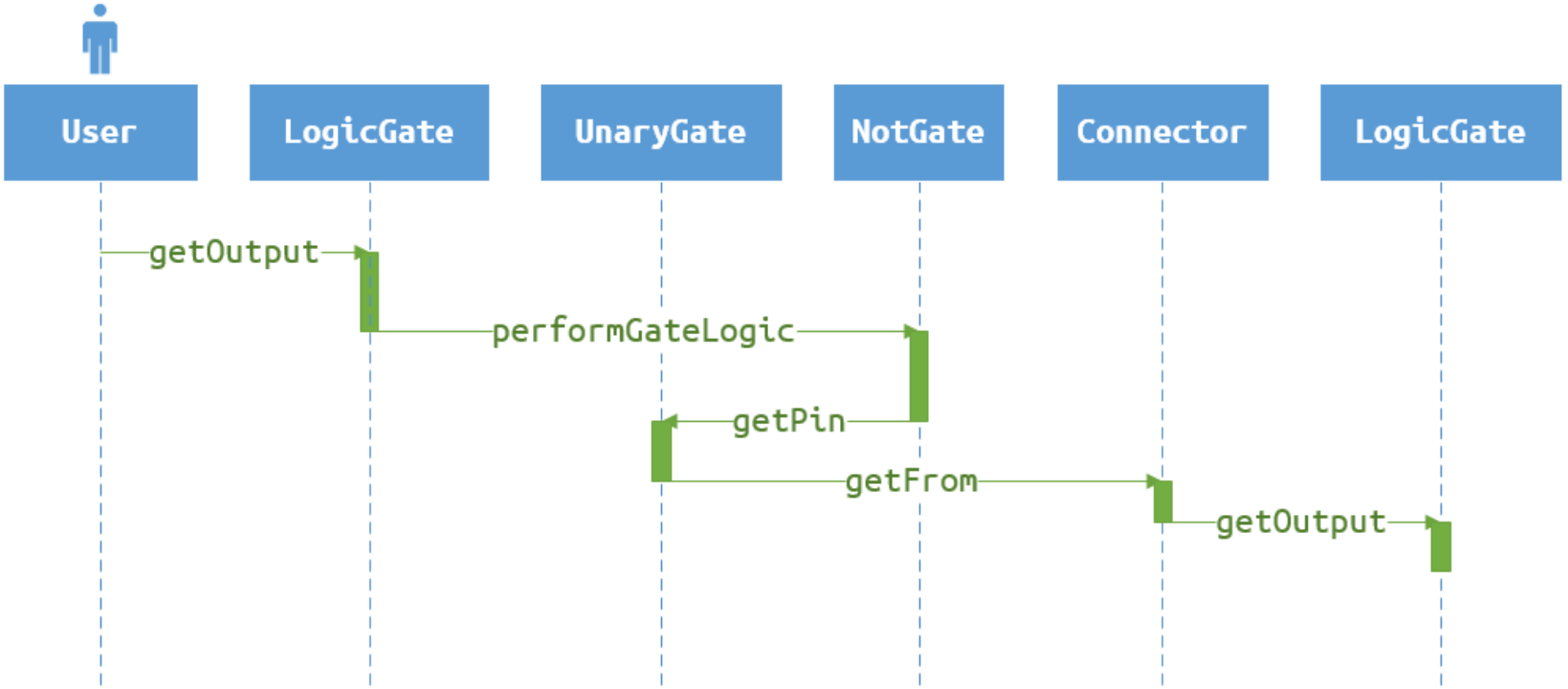
```
77
78  class Connector:
79
80      def __init__(self, fgate, tgate):
81          self.fromgate = fgate
82          self.togate = tgate
83          tgate.setNextPin(self)
84
85      def getFrom(self):
86          return self.fromgate
87
88      def getTo(self):
89          return self.togate
90
```

例子：组合逻辑电路

- UnaryGate实现
  - 利用之前未使用的状态pin表示输入端连接的Connector

```
13
14 class UnaryGate(LogicGate):
15     def setNextPin(self, source):
16         if self.pin == None:
17             self.pin = source
18         else:
19             raise RuntimeError('Error: no empty pins')
20     def getPin(self):
21         if self.pin == None:
22             return int(input("Enter Pin input for gate "
23                 + self.getLabel() + "-->"))
24         else:
25             return self.pin.getFrom().getOutput()
26
```

例子：组合逻辑电路


User
LogicGate
UnaryGate
NotGate
Connector
LogicGate
getOutput
performGateLogic
getPin
getFrom
getOutput

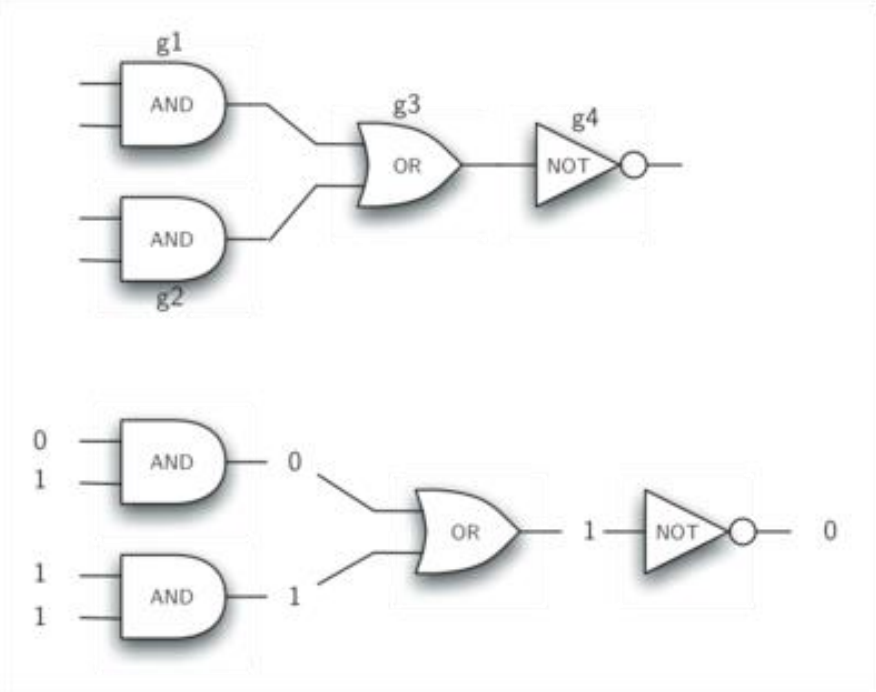
例子：组合逻辑电路

- BinaryGate实现
  - 同理，使用pinA和pinB

```
31
32  class BinaryGate(LogicGate):
33      def setNextPin(self, source):
34          if self.pinA == None:
35              self.pinA = source
36          elif self.pinB == None:
37              self.pinB = source
38          else:
39              raise RuntimeError('Error: no empty pins')
40      def getPinA(self):
41          if self.pinA == None:
42              return int(input("Enter Pin A input for gate "
43                  + self.getLabel() + "-->"))
44          else:
45              return self.pinA.getFrom().getOutput()
46
```

例子：组合逻辑电路

- 实现组合逻辑电路




```
>>> g1 = AndGate("G1")
>>> g2 = AndGate("G2")
>>> g3 = OrGate("G3")
>>> g4 = NotGate("G4")
>>> c1 = Connector(g1, g3)
>>> c2 = Connector(g2, g3)
>>> c3 = Connector(g3, g4)
>>> g4.getOutput()
Enter Pin A input for gate G1-->0
Enter Pin B input for gate G1-->1
Enter Pin A input for gate G2-->1
Enter Pin B input for gate G2-->1
0
```