AI 프로그래밍(파일,모듈,객체)

융합학과 권오영

oykwon@koreatech.ac.kr

파일과 모듈

파일

❖ 파일 핸들 제공: 운영체제에 무관하게 파일을 이 핸들을 통해서 다룸

nameHandle = open('kids', 'w')

w -> write, r -> read, a -> append

```
nameHandle = open('kids', 'w')
for i in range(2):
    name = raw_input('Enter name: ')
    nameHandle.write(name + '\n')
nameHandle.close()
```

```
nameHandle = open('kids', 'w')
nameHandle.write('Michael\n')
nameHandle.write('Mark\n')
nameHandle.close()
nameHandle = open('kids', 'r')
for line in nameHandle:
    print(line[:-1])
nameHandle.close()
```

It will print

```
Michael
Mark
```

```
nameHandle = open('kids', 'r')
for line in nameHandle:
    print(line)
nameHandle.close()
```

```
nameHandle = open('kids', 'a')
nameHandle.write('David\n')
nameHandle.write('Andrea\n')
nameHandle.close()
nameHandle = open('kids', 'r')
for line in nameHandle:
    print(line[:-1])
nameHandle.close()
```

it will print

```
Michael
Mark
David
Andrea
```

파일연산

open(fn, 'w') fn is a string representing a file name. Creates a file for writing and returns a file handle.

open(fn, 'r') fn is a string representing a file name. Opens an existing file for reading and returns a file handle.

open(fn, 'a') fn is a string representing a file name. Opens an existing file for appending and returns a file handle.

fh.read() returns a string containing the contents of the file associated with the file handle fh.

fh.readline() returns the next line in the file associated with the file handle fh.

fh.readlines() returns a list each element of which is one line of the file associated with the file handle fh.

fh.write(s) write the string s to the end of the file associated with the file handle fh.

fh.writeLines(S) S is a sequence of strings. Writes each element of S to the file associated with the file handle fh.

fh.close() closes the file associated with the file handle fh.

모듈

- 프로그램이 커지면 한 파일에 모든 내용을 담을 수 없다.
- 팀으로 프로그램을 작성할 경우 각자가 담당한 부분을 개별 파일로 관리해야 한다.
- 모듈: 파이썬 파일(.py)로 definitions과 statements로 구성
  - 단일 파일(M.py)로 구성
  - 모듈은 import M으로 현재 프로그램 영역을 불러들인다.
  - 모듈내의 객체를 접근하기 위해선 M.x 형태로 dot notation 사용

- 파일 circle.py

```
pi = 3.14159

def area(radius):
    return pi*(radius**2)

def circumference(radius):
    return 2*pi*radius

def sphereSurface(radius):
    return 4.0*area(radius)

def sphereVolume(radius):
    return (4.0/3.0)*pi*(radius**3)
```

모듈 활용

```
import circle
print (circle.pi)
print (circle.area(3))
print (circle.circumference(3))
print (circle.sphereSurface(3)
```

모듈

❖ 현재 프로그램의 scope 안에 모듈에 선언된 모든 객체를 동등하게 묶을(binding)수 있다.

from M import *

```
from circle import *
print pi
print circle.pi

NameError: name 'circle' is not defined
```

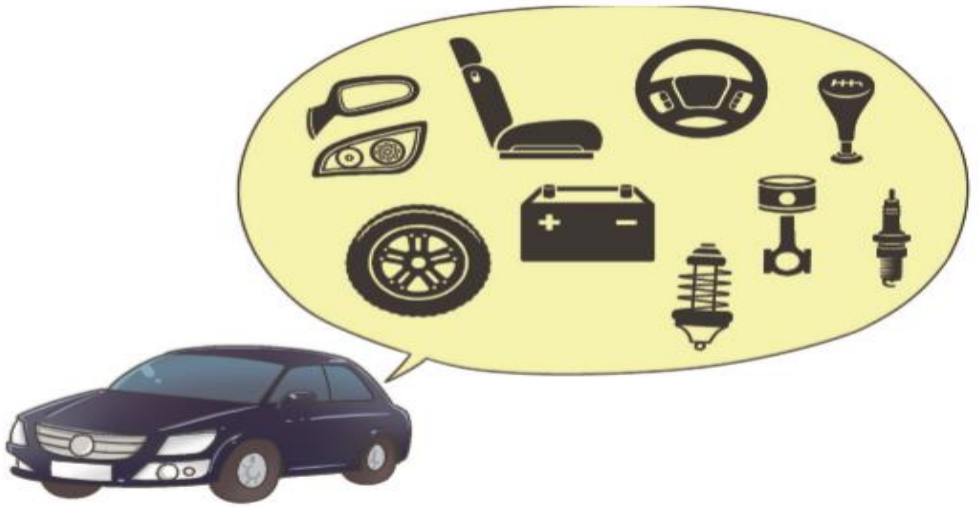
❖ 모듈안의 모든 값을 동등한 scope 수준으로 읽어들여서 처음 print pi는 3.14159를 출력하지만 두번째 print는 오류 발생.

❖ 모듈을 어떻게 사용할지는 프로그래머가 결정해야 한다.
- 내부에 미리 선언된 변수나 함수들과 충돌 여부를 확신할 수 있을까?

❖ 모듈을 import하고 중간에 모듈의 코드를 수정하면?
- 처음 import시에 로드되어서 중간에 수정된 코드는 반영되지 않음
- reload()를 수행하며 수정사항을 반영할 수 있다.

모듈

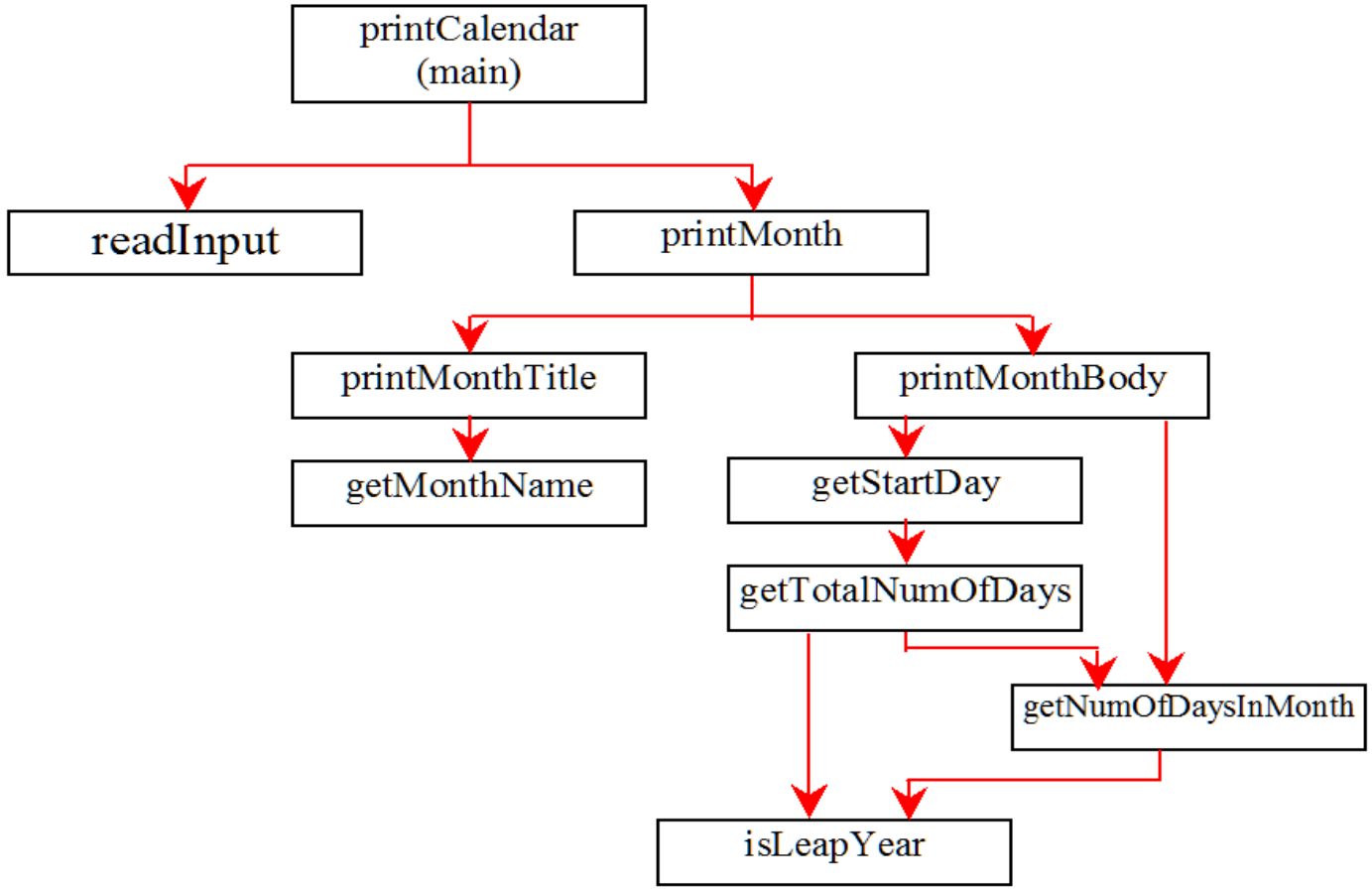
- 문제를 한 번에 해결하려고 하지 말고 더 작은 크기의 문제들로 분해한다. 문제가 충분히 작아질 때까지 계속해서 분해한다.
- 문제가 충분히 작아졌으면 각각의 문제를 모듈화한다
- 이들 모듈들을 조립하면 최종 프로그램이 완성된다.



Top-Down

❖ 적절히 pass keyword를 사용하여 top-down 개발에 사용

```
def printMonth(month)
    pass
( 지금은 pass 하고 나중에
 개발하겠음)
```



객체

출처:https://runestone.academy/runestone/books/published/pythonds3/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html

객체

❖ 사전적 정의

1. 주체로 부터 독립되어 있는 인간의 인식과 실천의 대상
2. 의사나 행위가 미치는 대상

❖ 컴퓨터 과학에서 **객체** 또는 **오브젝트**(object)는 클래스에서 정의한 것을 토대로 메모리(실제 저장공간)에 할당된 것으로 프로그램에서 식별자(변수)에 의해 참조되는 공간을 의미

- 객체는 클래스의 인스턴스
  (클래스는 틀(붕어빵틀), 객체는 틀을 바탕으로 만들어진 실체(붕어빵))

❖ 객체(object)는 어떠한 속성값과 행동을 가지고 있는 데이터

- 정수, 실수, 리스트등 각종 자료형도 모두 객체임

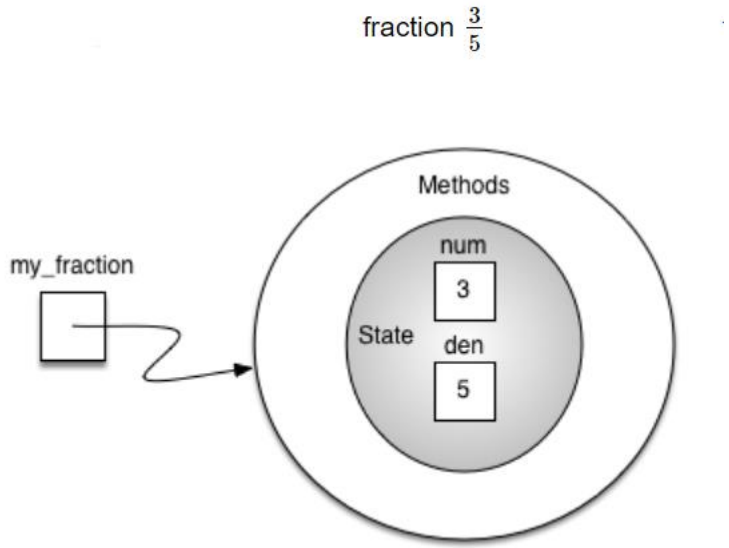
객체의 정의

❖ 클래스(class)를 사용하여 객체를 정의
- 객체지향언어마다 정의하는 문법이 다름
- Python은 class라는 예약어를 사용하고 클래스 명은 대문자로 시작

❖ 클래스를 이용하면 새로운 자료형들을 만들어 갈 수 있음
- 분수(fraction)를 다루어 보자.
  - ✓ 값(상태) : 분자, 분모
  - ✓ 행동: +, 비교(==), 출력(a/b 형태로 출력)

예) >>> my_fraction = Fraction(3,5)

```
>>> f1 = Fraction(1,4)
>>> f2 = Fraction(1,2)
>>> f3 = f1 + f2
>>> print(f3)
3/4
```

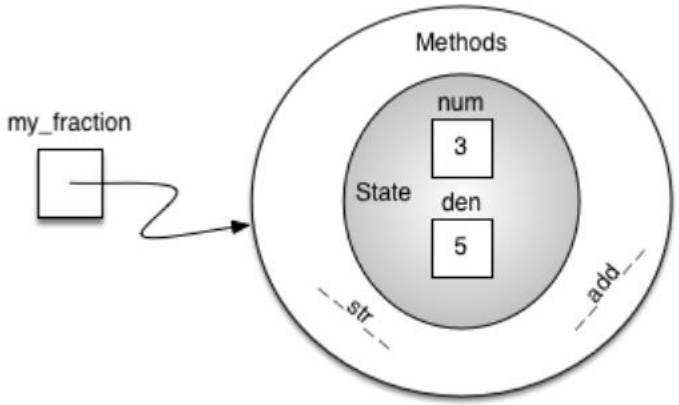


객체의 정의

❖ 클래스 Fraction (in Python)

참고: https://youtu.be/gFb9tvJZHXo

```
def gcd(m, n):
    while m % n != 0:
        m, n = n, m % n
    return n
```




```
class Fraction:
    """Class Fraction"""
    def __init__(self, top, bottom):
                """Constructor definition"""
        self.num = top
        self.den = bottom

    def __str__(self):
        return f"{self.num}/{self.den}"

    def __add__(self, other_fraction):
        new_num = self.num*other_fraction.den + self.den*other_fraction.num
        new_den = self.den*other_fractcion.den
        common = gcd(new_num, new_den)  ## 약분
        return Fraction(new_num//common, new_den//common)

    def __eq__(self, other_fraction):
        first_num = self.num * other_fraction.den
        second_num = other_fraction.num * self.den
        return first_num == second_num
```

초기화 및 상태값설정

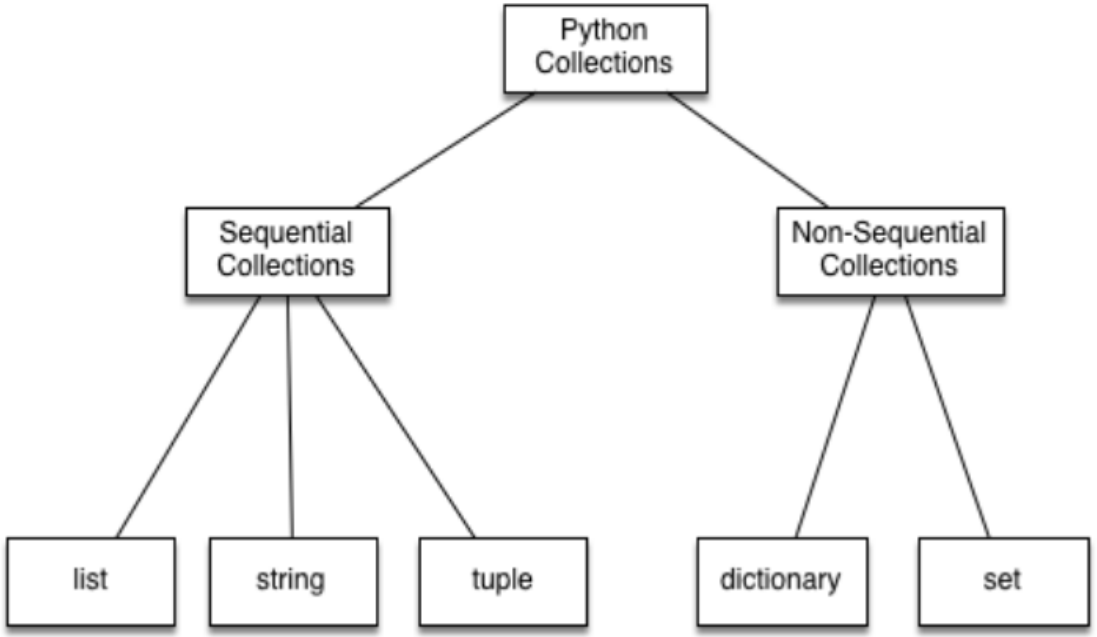
객체값을 출력하도록 만든 함수를 재정의 (a/b 형태출력)

더하기(+) 연산을 재정의

비교(==) 연산을 재정의

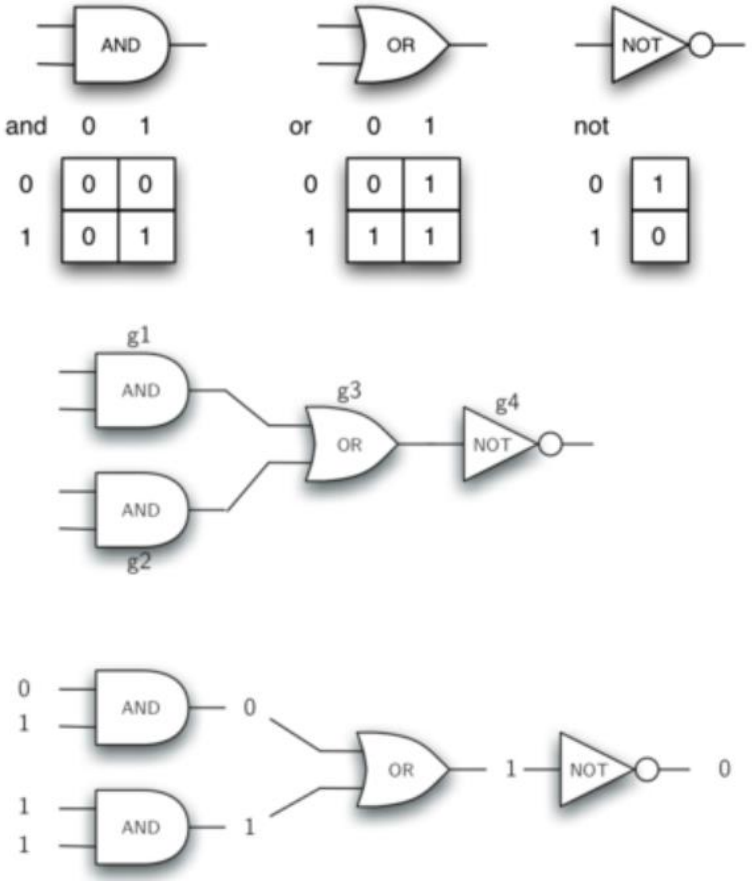
상속

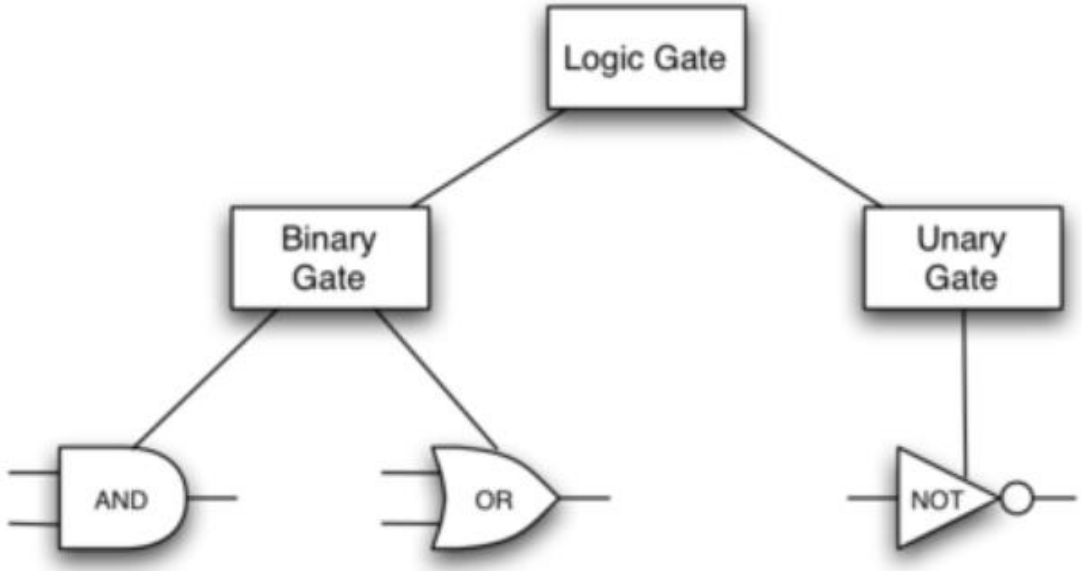
- 객체를 사용하면 상위객체에서 설정된 내용을 하위 객체가 상속받아 사용할 수 있다.
- **상속**(inheritance): 기본 클래스의 공통 기능을 물려받고, 다른 부분만 추가 또는 변경
  - 기본 클래스는 부모 클래스(또는 상위 클래스), Parent, Super, Base class 라고 부름
  - 기본 클래스 기능을 물려받는 클래스는 자식 클래스(또는 하위 클래스), Child, Sub, Derived class 라고 부름



Logic Gates의 연결

- 논리연산을 게이트들을 연결하여 논리 연산을 수행
- 논리 게이트를 클래스로 정의할 수 있음





KOREATECH 한국기술교육대학교

필요한 클래스 정의

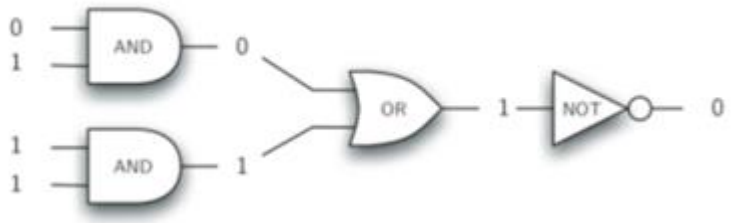
```
class LogicGate:

    def __init__(self, lbl):
        self.name = lbl
        self.output = None

    def get_label(self):
        return self.name

    def get_output(self):
        self.output = self.perform_gate_logic()
        return self.output
```

```
class BinaryGate(LogicGate):
    def __init__(self, lbl):
        super(BinaryGate, self).__init__(lbl)
        ## or LogicGate.__init__(lbl)
        self.pin_a = None
        self.pin_b = None
    def get_pin_a(self):
        if self.pin_a == None:
            return int(input("Enter pin A input for gate " + self.get_label() + ": "))
        else:
            return self.pin_a.get_from().get_output()
    def get_pin_b(self):
        if self.pin_b == None:
            return int(input("Enter pin B input for gate " + self.get_label() + ": "))
        else:
            return self.pin_b.get_from().get_output()
    def set_next_pin(self, source):
        if self.pin_a == None:
            self.pin_a = source
        else:
            if self.pin_b == None:
                self.pin_b = source
            else:
                print("Cannot Connect: NO EMPTY PINS on this gate")
```

필요한 클래스 정의

```
class AndGate(BinaryGate):

    def __init__(self, lbl):
        BinaryGate.__init__(self, lbl)

    def perform_gate_logic(self):
        a = self.get_pin_a()
        b = self.get_pin_b()
        if a == 1 and b == 1:
            return 1
        else:
            return 0
```

```
class OrGate(BinaryGate):

    def __init__(self, lbl):
        BinaryGate.__init__(self, lbl)

    def perform_gate_logic(self):
        a = self.get_pin_a()
        b = self.get_pin_b()
        if a == 1 or b == 1:
            return 1
        else:
            return 0
```




```
>>> g1 = AndGate("G1")
>>> g1.get_output()
Enter pin A input for gate G1: 1
Enter pin B input for gate G1: 0
0

>>> g2 = OrGate("G2")
>>> g2.get_output()
Enter pin A input for gate G2: 1
Enter pin B input for gate G2: 1
1
```

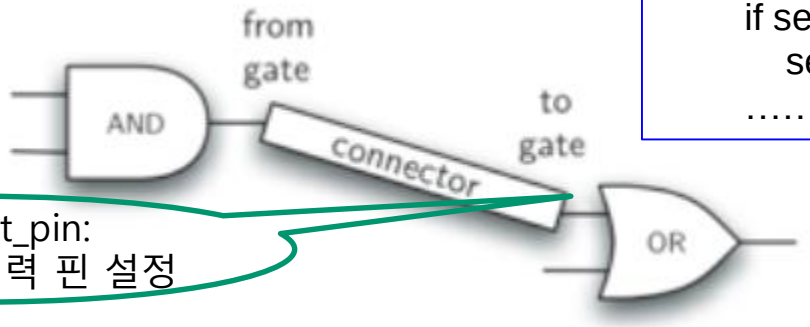
필요한 클래스 정의

```
class UnaryGate(LogicGate):

    def __init__(self, lbl):
        LogicGate.__init__(self, lbl)

        self.pin = None

    def get_pin(self):
        if self.pin == None:
            return int(input("Enter pin input for gate " + self.get_label() + ": "))
        else:
            return self.pin.get_from().get_output()

    def set_next_pin(self, source):
        if self.pin == None:
            self.pin = source
        else:
            print("Cannot Connect: NO EMPTY PINS on this gate")
```

```
class NotGate(UnaryGate):

    def __init__(self, nlbl):
        UnaryGate.__init__(self, lbl)

    def perform_gate_logic(self):
        if self.get_pin():
            return 0
        else:
            return 1
```

필요한 클래스 정의

```
class Connector:

    def __init__(self, fgate, tgate):
        self.from_gate = fgate
        self.to_gate = tgate

        tgate.set_next_pin(self)

    def get_from(self):
        return self.from_gate

    def get_to(self):
        return self.to_gate
```



set_next_pin:
OR 게이트 입력 핀 설정

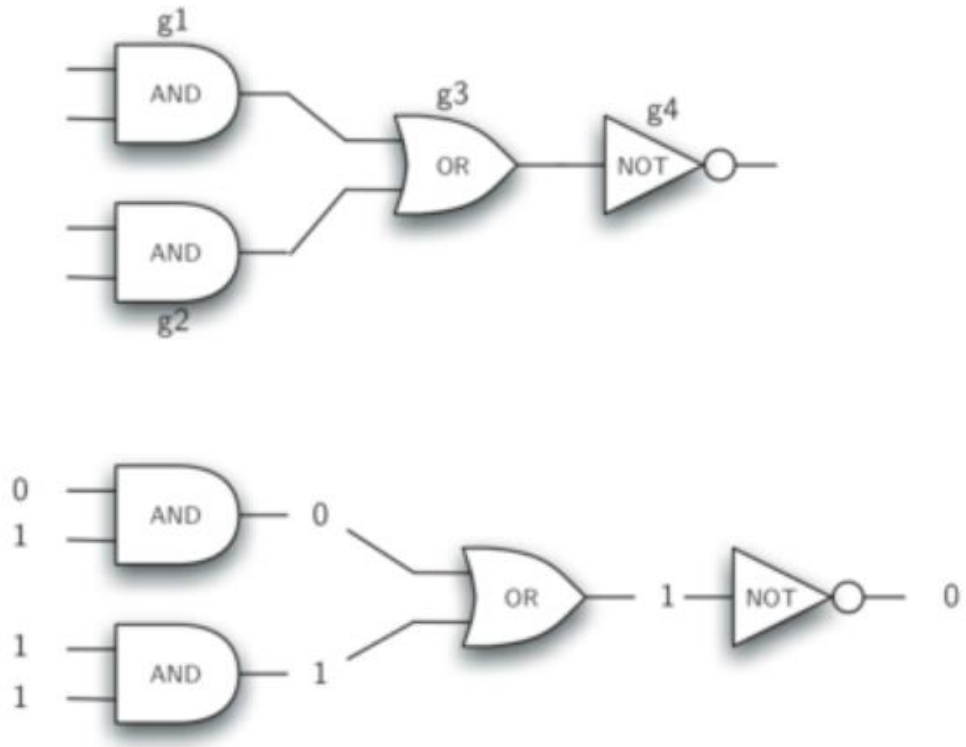
```
def set_next_pin(self, source):
        if self.pin_a == None:
            self.pin_a = source
        ……
```

```
def get_pin_a(self):
        if self.pin_a == None:
            return int(input("Enter pin A input for gate " + self.get_label() + ": "))
        else:
            return self.pin_a.get_from().get_output()
```

게이트의 입력을 직접 받거나 커넥터로 부터 받는다.
self.pin_a : 커넥터
self.pin_a.get_from : from gate
self.pin_a.get_from().get_output() : from gate 출력

결국 from gate의 출력이 to gate 입력으로 설정

Circuit 구성




```
def main():
    g1 = AndGate("G1")
    g2 = AndGate("G2")
    g3 = OrGate("G3")
    g4 = NotGate("G4")
    c1 = Connector(g1, g3)
    c2 = Connector(g2, g3)
    c3 = Connector(g3, g4)
    print(g4.get_output())

main()
```

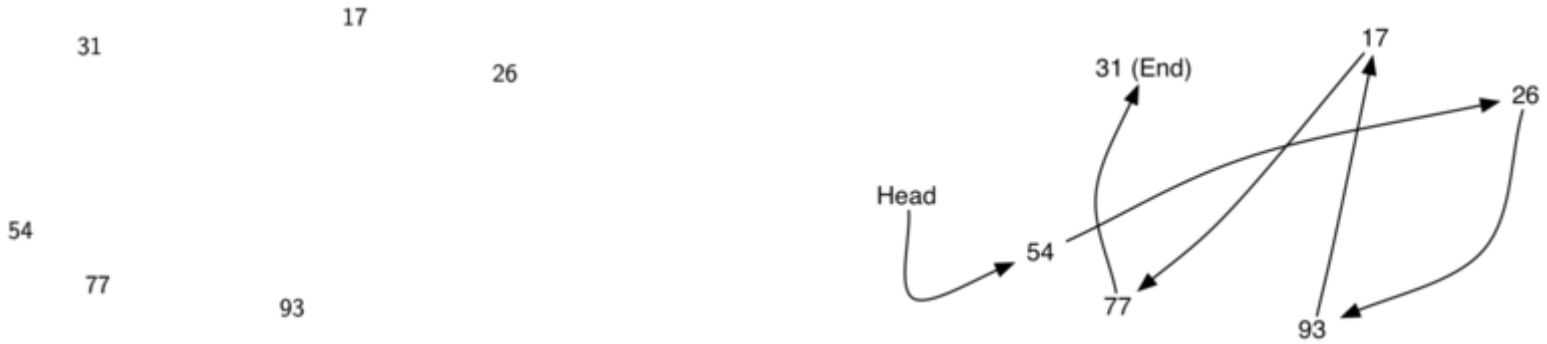
참고: https://youtu.be/brrpvAlzOyM

리스트 (LINKED LIST)

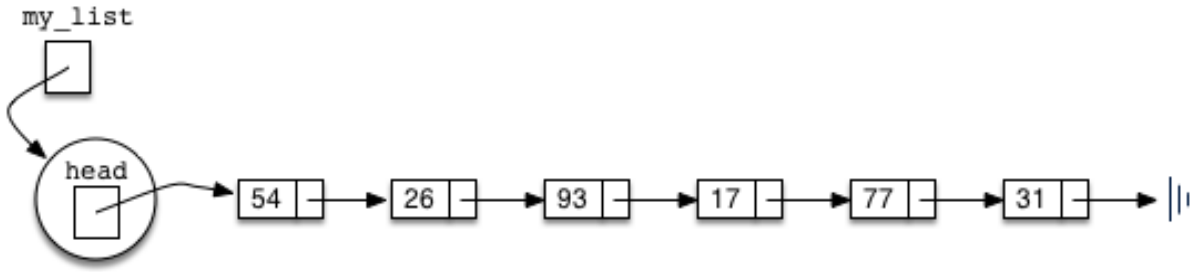
출처: https://runestone.academy/runestone/books/published/pythonds3/BasicDS/toctree.html

리스트 (linked list)

❖ 주어진 데이터들을 연결하여 관리



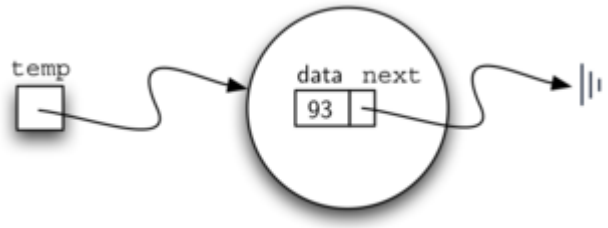
❖ Data와 link로 구성된 노드를 만들고, 이들 노드를 연결하여 리스트를 생성



리스트(linked list)

❖ 리스트에 사용할 노드의 선언

- >>> temp = Node(93)




```
class Node:
    """A node of a linked list"""

    def __init__(self, node_data):
        self._data = node_data
        self._next = None

    def get_data(self):
        return self._data

    def set_data(self, node_data):
        self._data = node_data

    def get_next(self):
        return self._next

    def set_next(self, node_next):
        self._next = node_next

    def __str__(self):
        return str(self._data)
```

❖ 노드 클래스만 사용한 리스트 프로그래밍

```
def add(data):
    node = head
    while node.next:
        node = node.next
    node.next = Node(data)

node1 = Node(1)
head = node1
for index in range(2, 10):
    add(index)

node = head
while node.next:
    print(node.data)
    node = node.next
print (node.data)
```

리스트 (linked list)

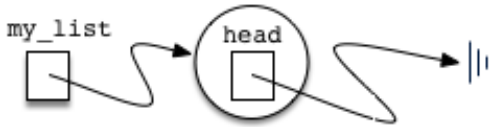
❖ 리스트의 연산 (파이선 리스트 연산과 유사)

- List() creates a new list that is empty. It needs no parameters and returns an empty list.
- add(item) adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.
- remove(item) removes the item from the list. It needs the item and modifies the list.
  Raise an error if the item is not present in the list.
- search(item) searches for the item in the list. It needs the item and returns a boolean value.
- is_empty() tests to see whether the list is empty. It needs no parameters and returns a boolean value.
- size() returns the number of items in the list. It needs no parameters and returns an integer.
- append(item) adds a new item to the end of the list making it the last item in the collection.
  It needs the item and returns nothing. Assume the item is not already in the list.
- index(item) returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
- insert(pos, item) adds a new item to the list at position pos. It needs the item and returns nothing.
  Assume the item is not already in the list and there are enough existing items to have position pos.
- pop() removes and returns the last item in the list. It needs nothing and returns an item.
  Assume the list has at least one item.
- pop(pos) removes and returns the item at position pos. It needs the position and returns the item.
  Assume the item is in the list.

리스트(linked list)

❖ 정렬되지 않은 리스트 클래스 선언

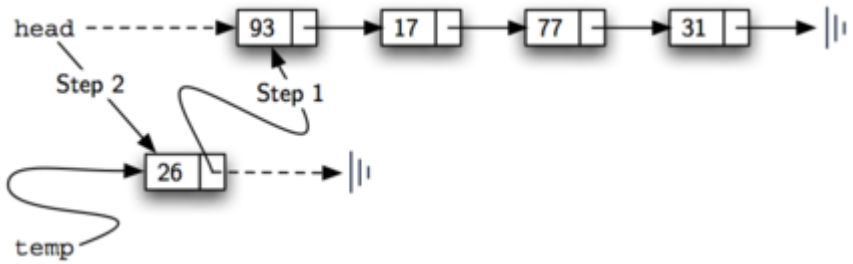
```
class UnorderedList:

    def __init__(self):
        self.head = None
```



- >>> my_list = UnorderedList()

❖ Data를 추가하는 add 연산

- 항상 헤드가 가리키는 곳에 추가한다.
- 31, 77, 17, 93을 추가했고,
  26을 추가하려한다.

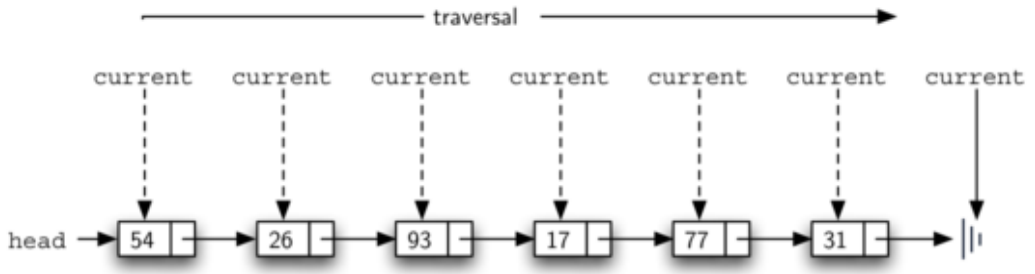



```
class UnorderedList:

    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head == None

    def add(self, item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp
```

리스트(linked list)

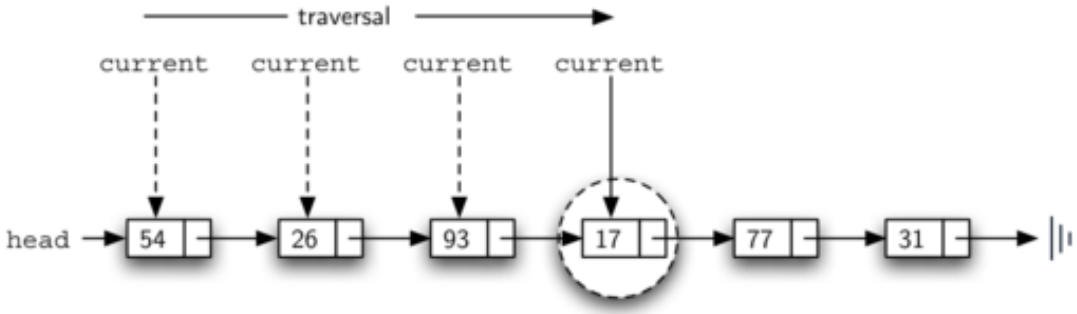
❖ UnorderedList를 위한 추가 연산 (size, search)

```
def size(self):
    current = self.head
    count = 0
    while current is not None:
        count = count + 1
        current = current.next

    return count
```




```
def search(self, item):
    current = self.head
    while current is not None:
        if current.data == item:
            return True
        current = current.next

    return False
```

```
>>> my_list.search(17)
```

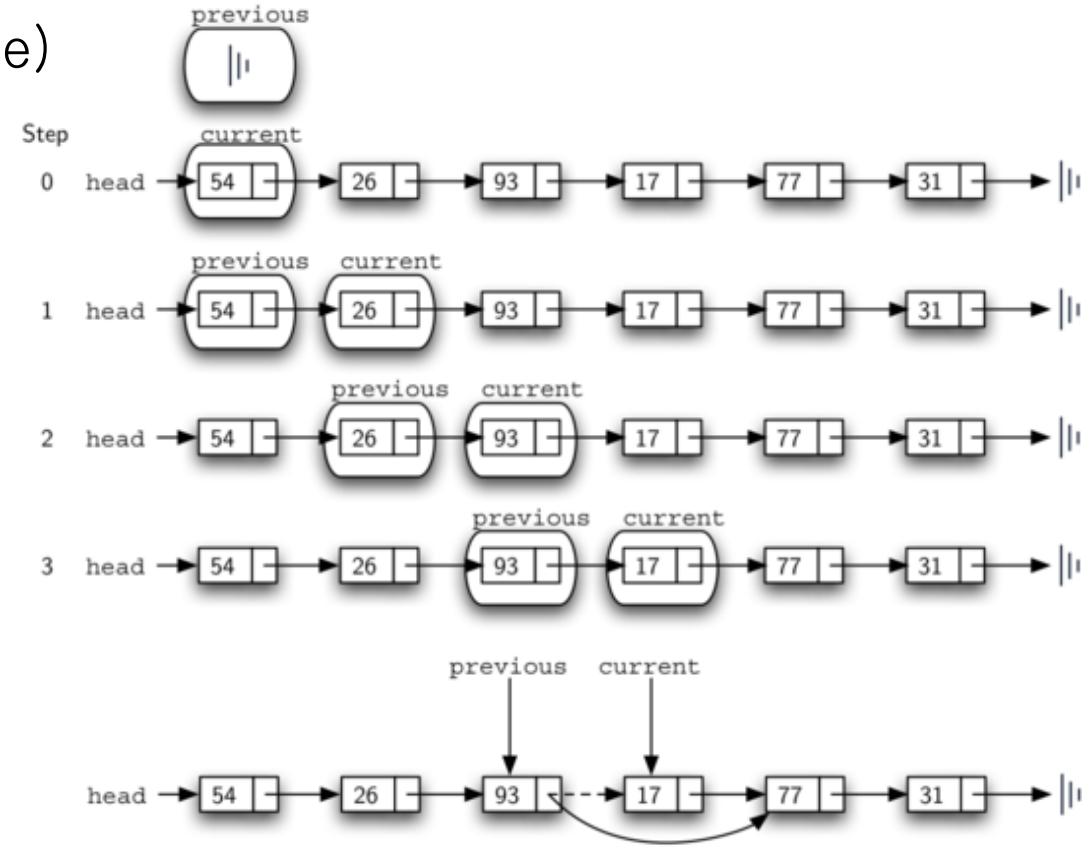


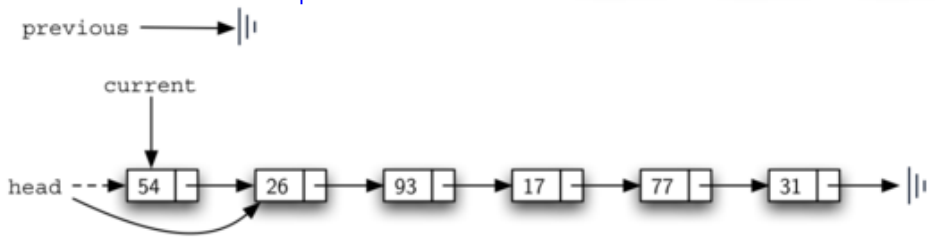
리스트(linked list)

❖ UnorderedList를 위한 추가 연산 (remove)

- 리스트 상의 17을 제거

```
def remove(self, item):
    current = self.head
    previous = None

    while current is not None:
        if current.data == item:
            break
        previous = current
        current = current.next

    if current is None:
        raise ValueError("{} is not in the list".format(item))
    if previous is None:
        self.head = current.next
    else:
        previous.next = current.next
```



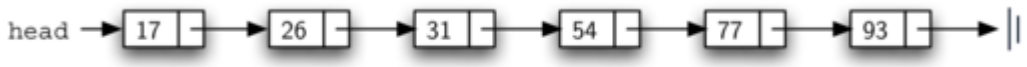


리스트(linked list)

❖ 정렬된 상태로 유지되는 리스트

- 데이터를 삽일 할 때 마다 정렬될 수 있도록 위치를 찾아서 데이터를 삽입

```
class OrderedList:

    def __init__(self):
        self.head = None
```

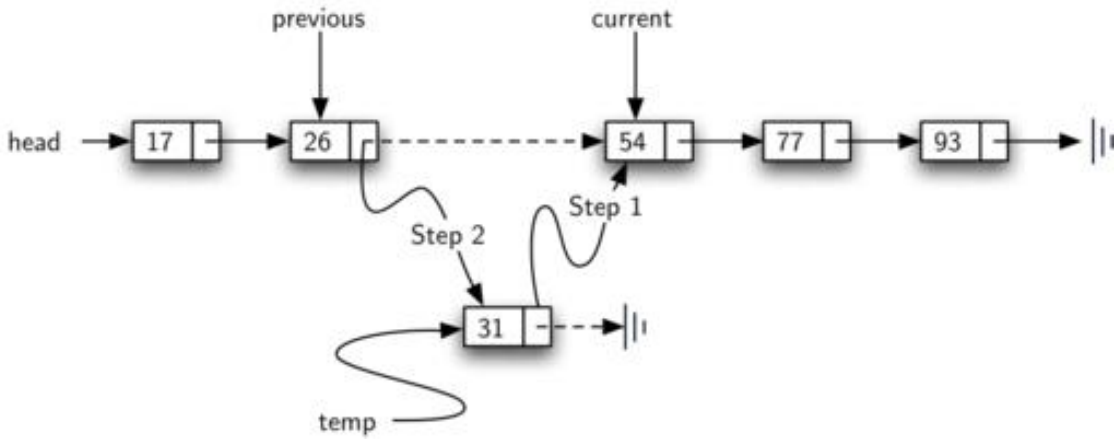


- OrderedList() creates a new ordered list that is empty.
- add(item) adds a new item to the list making sure that the order is preserved. It needs the item and returns nothing. Assume the item is not already in the list.
- remove(item)
- search(item)
- is_empty()
- size()
- index(item)
- pop()
- pop(pos)

리스트(linked list)

❖ 리스트가 17, 26, 54, 77, 93으로 구성되고, 31을 추가

```
class OrderedList:

    def __init__(self):
        self.head = None

    def add(self, item):
        """Add a new node"""
        current = self.head
        previous = None
        temp = Node(item)

        while current is not None and current.data < item:
            previous = current
            current = current.next

        if previous is None:
            temp.next = self.head
            self.head = temp
        else:
            temp.next = current
            previous.next = temp
```



Linked list는 배열(기본자료형 리스트)보다 삽입과 삭제가 쉽다.