

# 24. 해결책 모델링, ₩ 파일 입출력, ₩ 예외 처리, 중간과제

---

2018.12

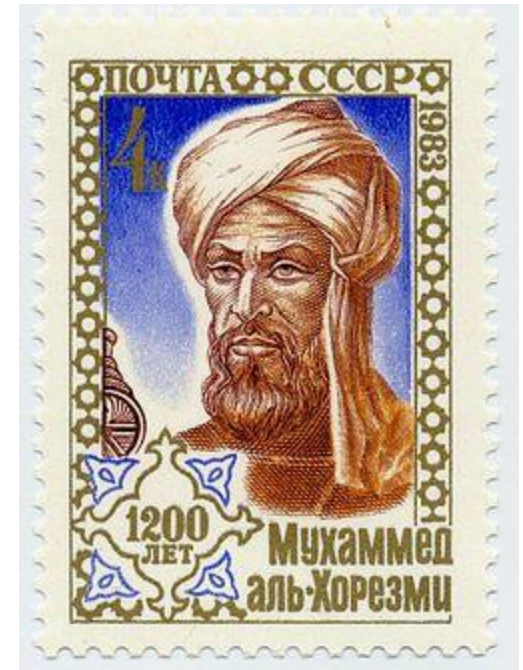
일병 김재형

# 알고리즘

---

## 알고리즘(algorithm)

- 목적을 달성하거나 문제해결을 하도록 만드는 독립된 동작들의 순서
- 페르시아의 수학자 - 알콰리즈미



# 알고리즘

---

## 알고리즘(algorithm)-컴퓨터

- 어떤 입력을 원하는 출력으로 변환하는 컴퓨터 동작의 순서
- Ex) 쿠키 요리법
  - 입력: 재료
  - 알고리즘: 요리법
  - 출력: 쿠키

- 1 컵 분량의 버터 녹인 것
- 컵 분량의 갈색 설탕
- 계란 2개
- 3 컵 분량의 밀가루
- 1 티스푼 분량의 베이킹 파우더
- 1 티스푼 분량의 베이킹 소다
- 2 2 컵 분량의 초콜렛 칩

1. 오븐을 화씨 375도로 미리 데운다.
2. 양피지 종이로 만든 쿠키 용지를 준비한다.
3. 한 용기에 버터, 갈색 설탕, 계란을 넣고 함께 젓는다.
4. 다른 용기에 밀가루, 베이킹 파우더와 베이킹, 소다를 넣고 섞는다. 점차적으로 설탕 혼합물을 섞는다.
5. 용기에 초콜렛 칩을 넣는다.
6. 한 스푼 용량의 쿠키 반죽으로 쿠키 용지를 채운다.
7. 9분간 오븐에 굽는다.
8. 쿠키 용지를 오븐에서 꺼내어 5분간 상온에서 식힌다.

# 알고리즘

---

## 알고리즘(algorithm)-4대요소

- 모든 동작에는 의미(semantic)를 가진다.
- 모든 동작은 모호하지 않다(unambiguous)
  - 다른 사람이 보아도 똑같이 해석된다.
- 동작의 수행 순서가 확실히 정의된다.
- 유한한 숫자의 동작을 실행 후 종료(halt)된다.

# 모델링

---

## 모델링이란?

- 추상화 과정을 통해 불필요한 세부내용을 제거하여 어떤 사물 또는 객체에서 중요한 특성을 강조하는 것

## 알고리즘의 모델링

- 활동 다이어그램 - 명령어(활동)
- 상태 다이어그램 - 데이터(상태)

# 활동 다이어그램

## 제어흐름

- 명령어(활동) 실행의 대략적인 순서(알고리즘)를 기술하기 위한 수단

## 활동 다이어그램

- 제어흐름을 순서도로 나타낸 것
- 즉, 알고리즘을 나타낸다.
- 추상화되어 표현되나 너무 모호하지 않아야 한다.



그림 6.1 활동 다이어그램에 사용되는 기호

# 활동 다이어그램

예시



그림 6.2 아침 일상에 대한 활동 다이어그램

# 활동 다이어그램

## 선택

- 실행 중 선택을 해야할 경우
- 조건문을 기반으로 발생
- 분기로 표현한다.

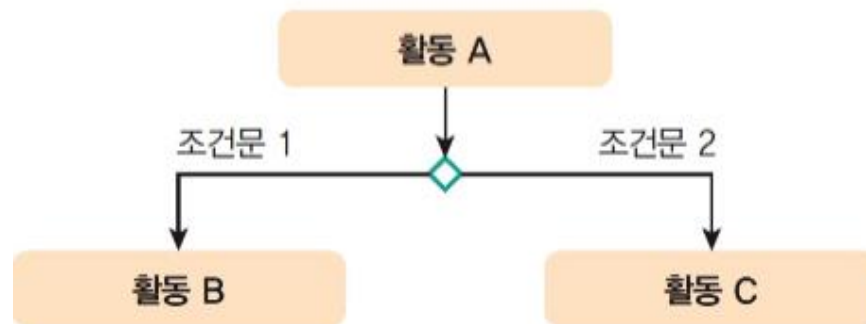


그림 6.3 선택에 결정 기호를 사용하기



# 활동 다이어그램

## 선택

- 다이아몬드 기호를 사용
- 결정 다이아몬드
  - 조건이 꼬리표로 붙는다.
- 병합 다이아몬드
  - 병합하는 결정이 있어 꼬리표가 없다

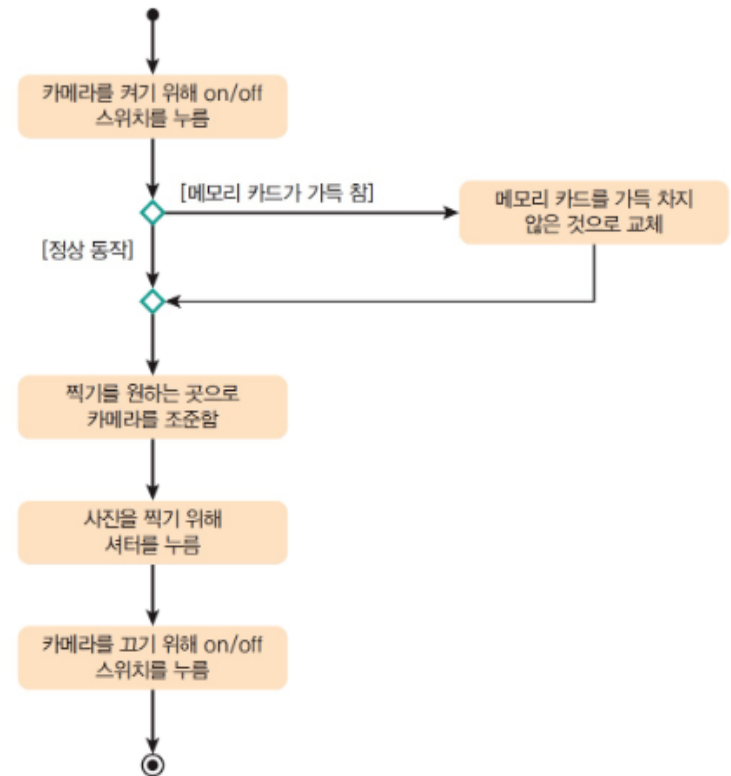
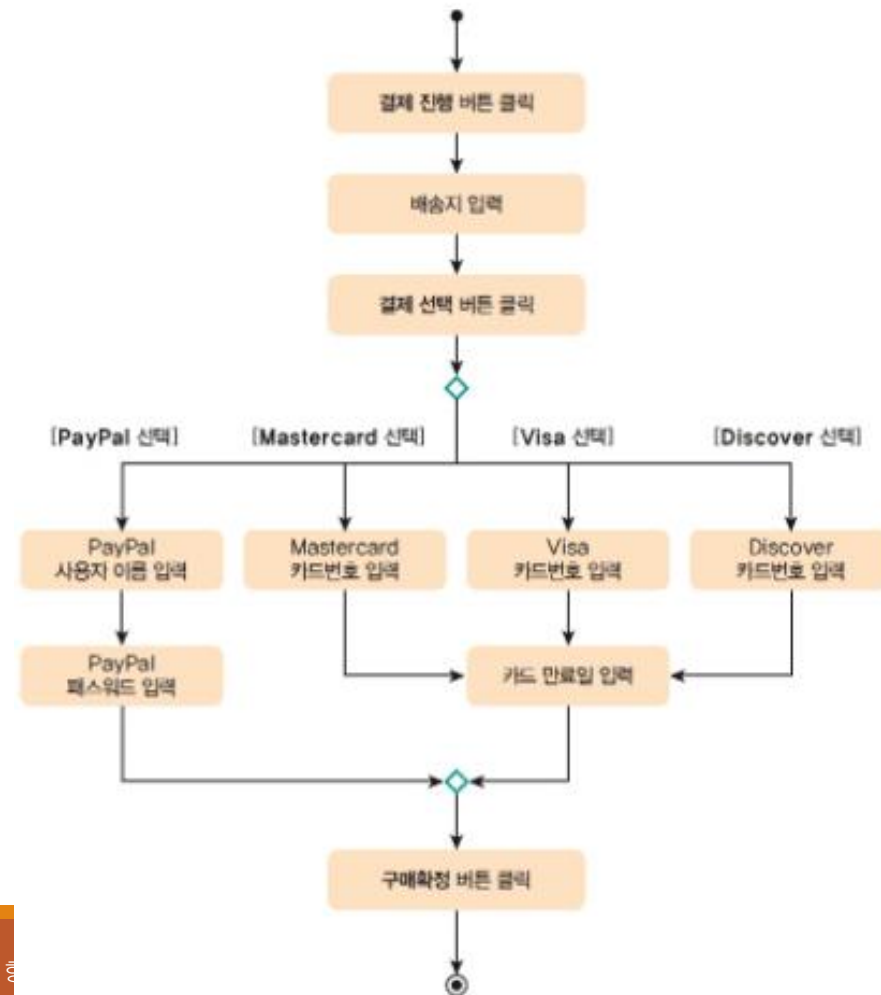


그림 6.5 카메라 사용에 대한 활동 다이어그램

# 활동 다이어그램

## 선택

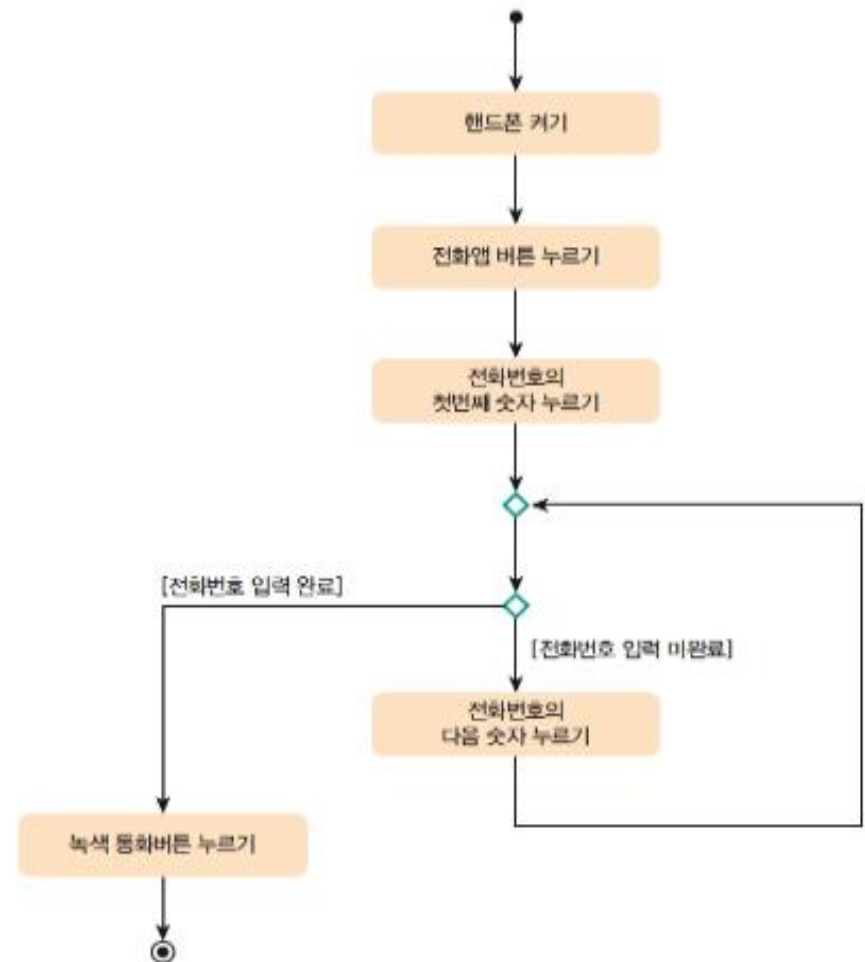
- 많은 선택상황을 다룰 수 있다.



# 활동 다이어그램

## 반복

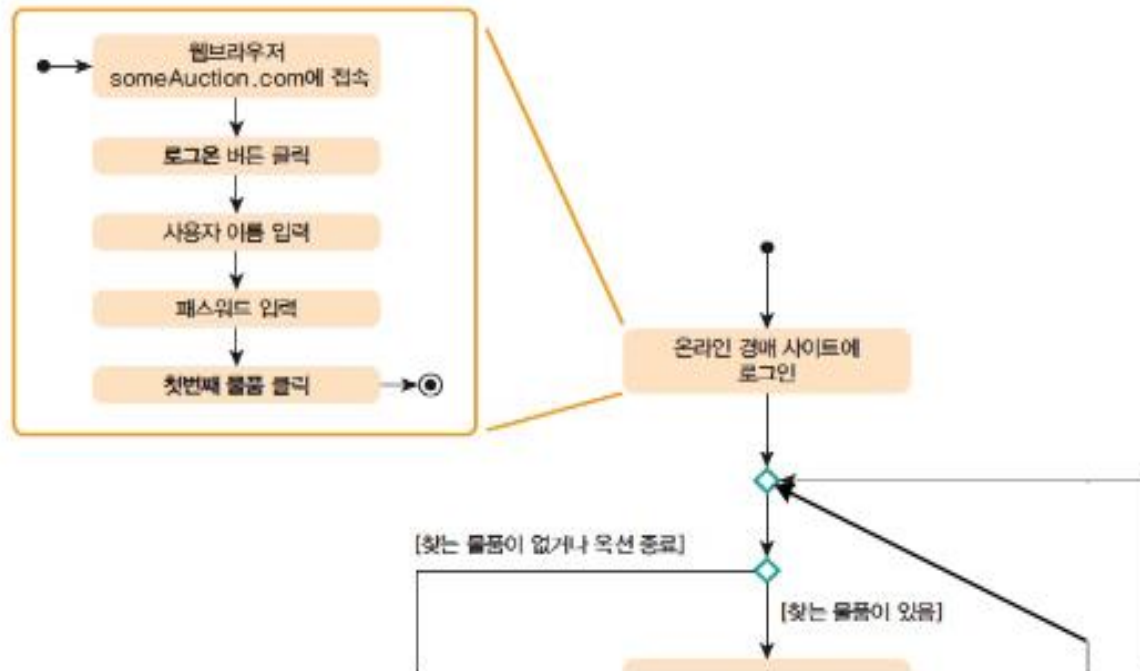
- 이전 단계로 돌아오는 화살표로 표현
- 루프를 형성



# 활동 다이어그램

## 추상화

- 복잡한 활동을 사각형으로 추상화
- 세부적으로 분해한다.



# 상태 다이어그램

---

## 상태(State)

- 소프트웨어의 상태
- ex) 온도 조절 시스템
  - 꺼짐, 난방, 냉방
- ex) 물의 상태
  - 고체, 액체, 기체 => 간단하게

# 상태 다이어그램

---

## 상태(State)

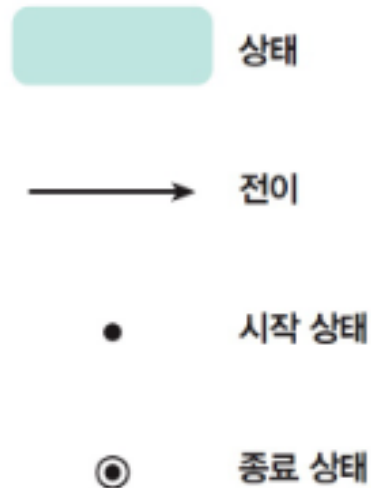
- 항공사 홈페이지
  - 운항 정보를 확인하기 위한 상태
  - 예약 갱신을 하기 위한 상태
  - 개인 계정을 확인 및 변경을 하기 위한 상태
- 사용자는 각 상태 별로 서로 다른 작업을 한다.

# 상태 다이어그램

---

## 상태 다이어그램

- 소프트웨어의 상태가 변화하는 것을 나타낸 다이어그램
- 상태 다이어그램의 기호

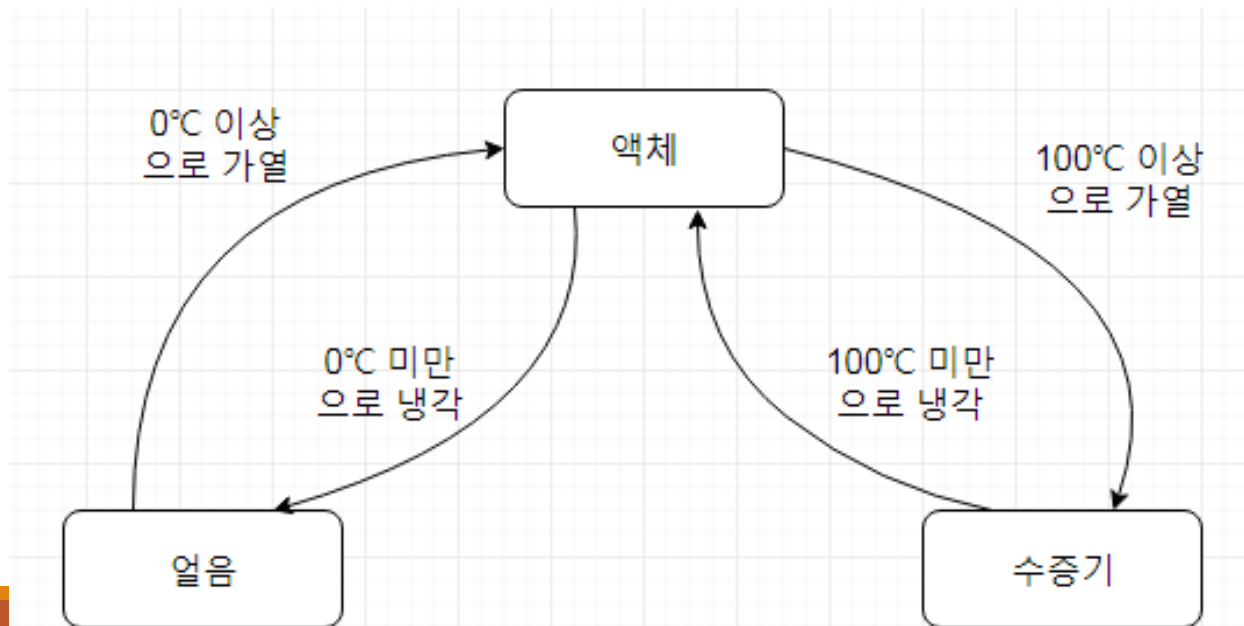


**그림 6.10** 상태 다이어그램의 표기 기호

# 상태 다이어그램

## 상태 다이어그램

- 이벤트(event)
  - 조건이나 주변 환경을 의미
  - 이벤트가 발생할 경우만 상태의 전이가 일어난다





# 상태 다이어그램

## 상태 다이어그램

- 복잡한 다이어그램
- 같은 상태로도 전이가 된다.

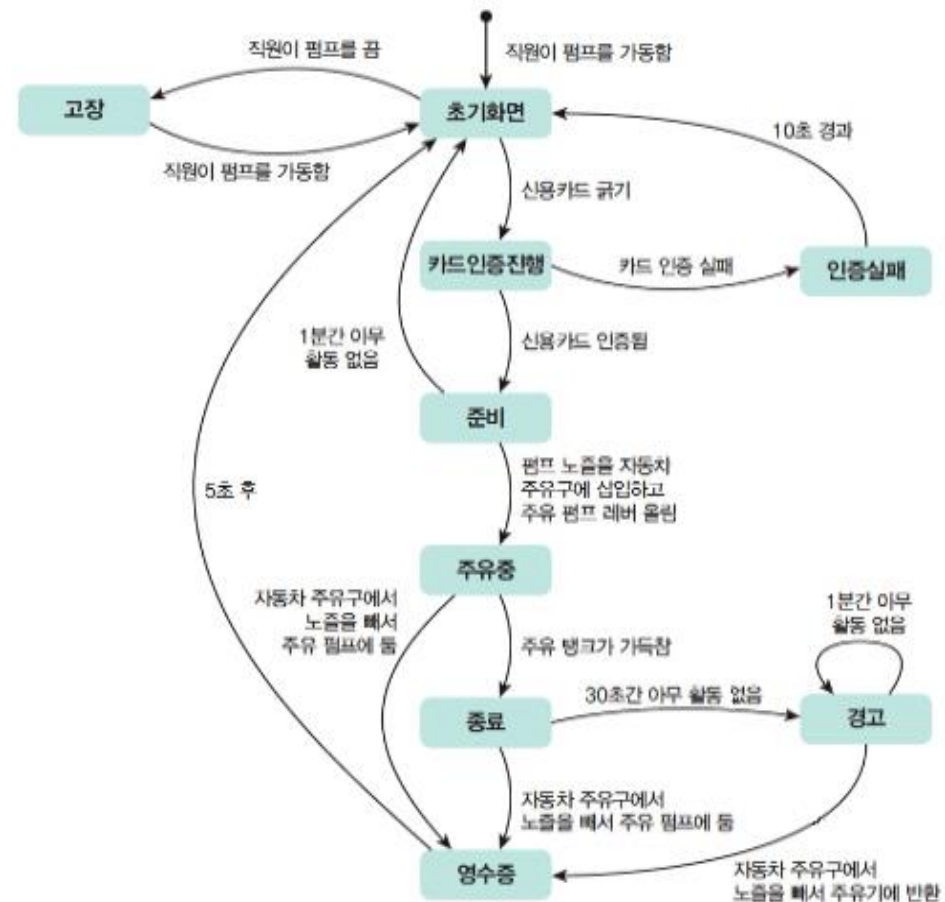


그림 6.12 주유소 펌프 상태 다이어그램

# 상태 다이어그램

## 상태 다이어그램에 행동 포함

- entry
  - 상태로 전이되었을 때 필요한 동작
- do
  - 상태에 머물러 있는 동안 계속 실행되는 동작
- exit
  - 다른 상태로 전이 될 때 필요한 동작

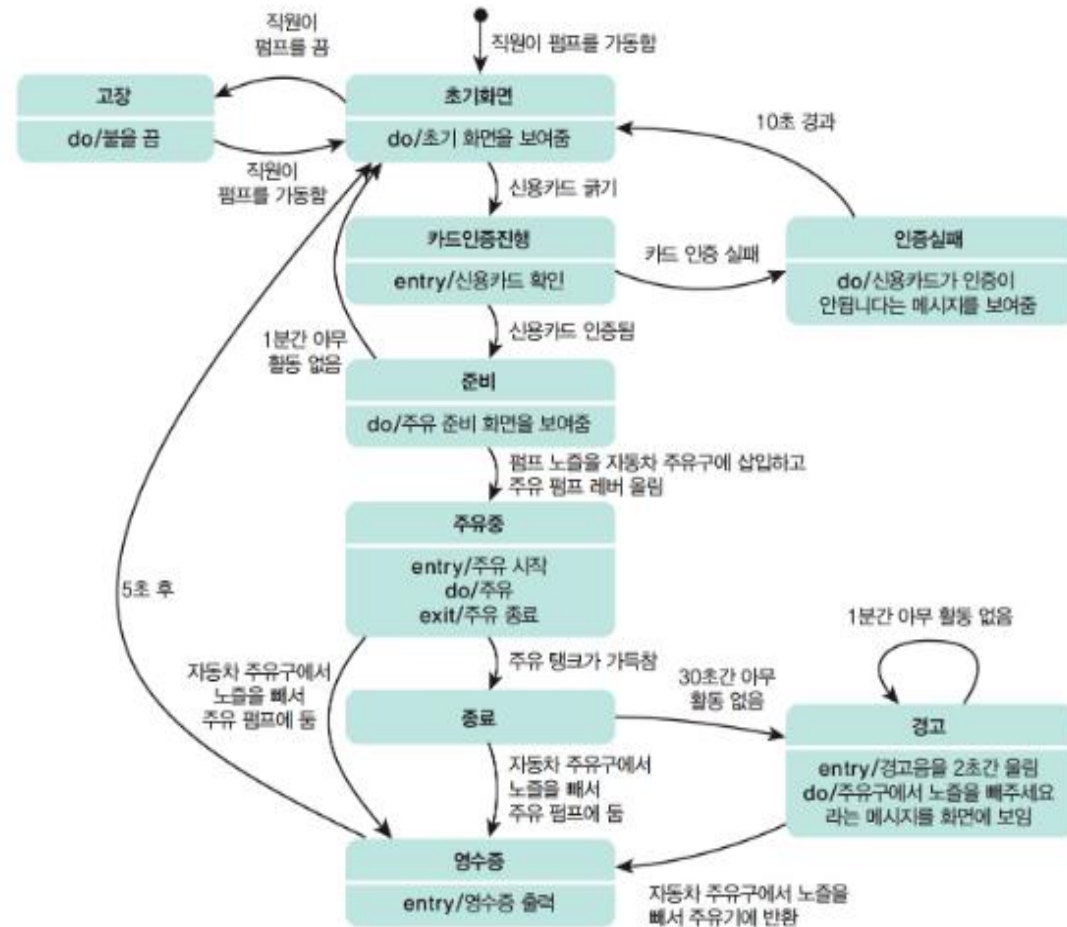


그림 6.13 액션을 추가한 주유소 펌프 상태 다이어그램

# 상태 다이어그램

---

## 상태 다이어그램의 상세정보

- 상태 다이어그램을 설계 시
  - 1) 문제를 단순화
  - 2) 분할 정복을 사용
- 종이 한 장이나 컴퓨터 화면 하나에 다 그려지는 것이 좋다.

# 상태 다이어그램

## 상태 다이어그램의 상세정보

- +/-와 충전 중단이 정확히 기술되지 않음
- +/- 누르면 재생으로 전이 하지만 동작을 모른다.
- 충전 중 뽐으면?

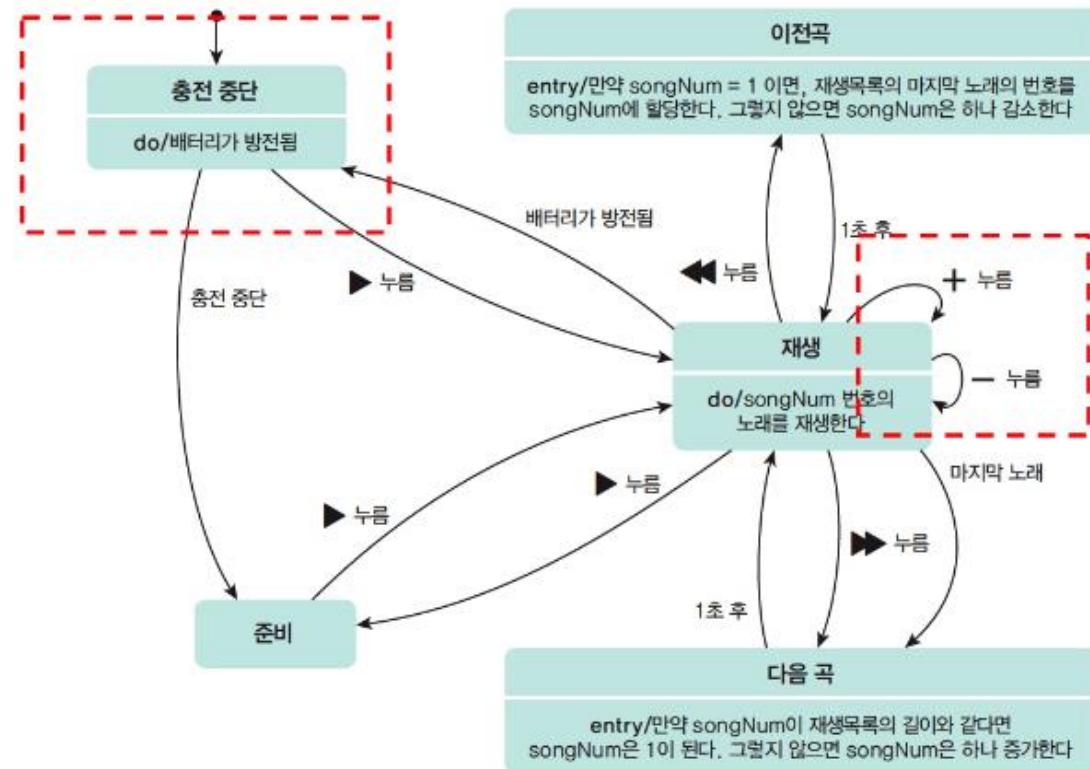


그림 6.16 뮤직 플레이어의 상태 다이어그램

## 상태 다이어그램의 상세정보

- 
- The diagram illustrates the state transitions for a music player system. It is divided into three main sections: **배터리** (Battery), **이전곡** (Previous Song), and **다음곡** (Next Song).
- 배터리 (Battery) States:**
- 방전배터리 (Discharged Battery):**
    - entry/비프음을 세번 울린다
    - exit/songNum에 1을 할당한다
  - 충전 (Charging):**
    - entry/배터리 충전: 충전 LED가 켜짐
  - 충전됨 (Charged):**
    - do/충전 LED가 녹색이 된다
- 이전곡 (Previous Song) States:**
- 재생 (Play):**
    - 음량줄이기 (Volume Down):
      - entry/음량을 낮춘다
    - 정상 (Normal):
      - entry/음량을 낮춘다
    - 음량낮추기 (Volume Up):
      - entry/음량을 낮춘다
  - songNum 변수의 노래를 재생한다**
- 다음곡 (Next Song) States:**
- entry/만약 songNum 이 재생목록의 길이와 같다면 songNum은 1이 된다. 그렇지 않으면 songNum은 하나 증가한다**
- Transitions:**
- 충전 (Charge):** From **방전배터리** to **충전**.
  - 충전됨 (Charged):** From **충전** to **충전됨**.
  - 충전 중단 (Charge Stop):** From **충전됨** to **준비**.
  - 준비 (Ready):** From **충전됨** to **준비**.
  - 재생 (Play):** From **준비** to **재생**.
  - 다음곡 (Next Song):** From **재생** to **다음곡**.
  - 이전곡 (Previous Song):** From **다음곡** to **이전곡**.
  - 이전곡 (Previous Song):** From **이전곡** to **준비**.

201

# 상태 다이어그램

## 상태 다이어그램의 상세정보

- 외부상태: 확장된 상태 그 자체(배터리 방전, 재생)
- 내부상태: 확장된 상태 내부에 존재하는 상태

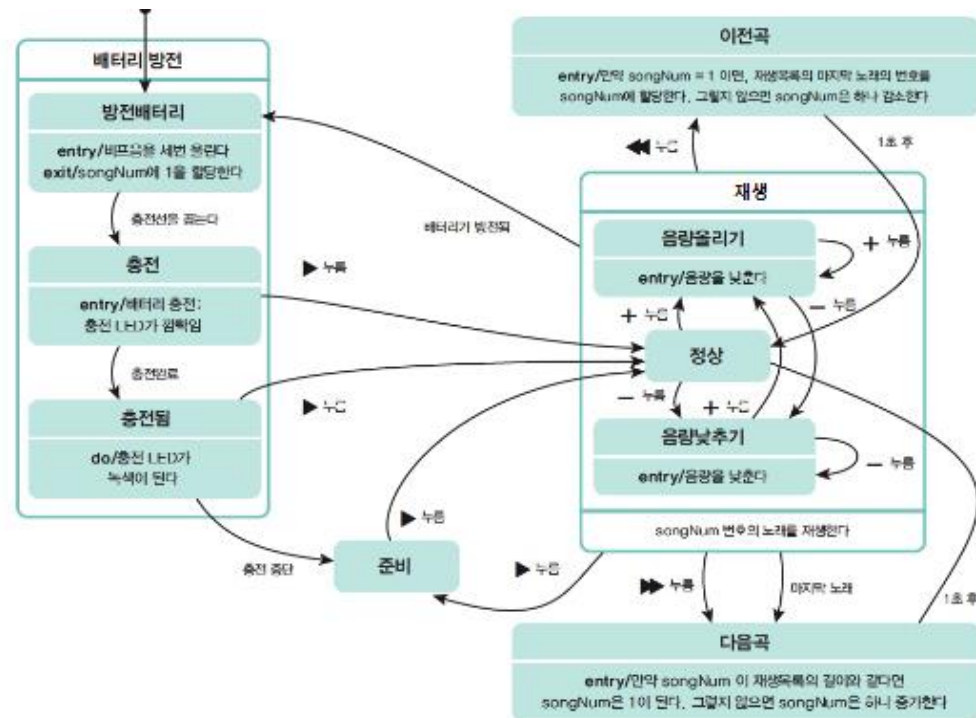
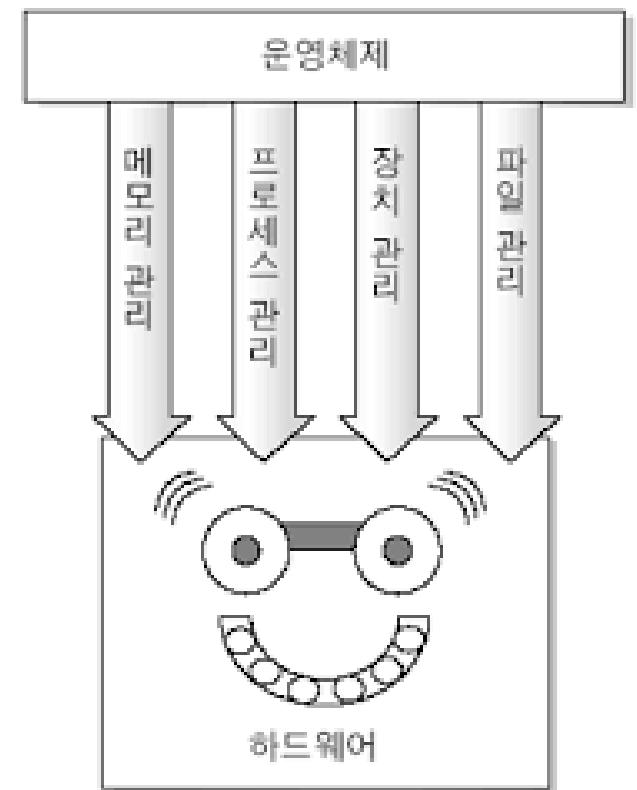


그림 6.17 뮤직 플레이어의 확장된 상태 다이어그램

# 파일 입/출력

## 파일 입/출력 기초

- 일반적인 프로그램이 하드웨어(하드디스크 등)를 직접 제어해 파일에 접근하는 경우는 매우 드물다.
- 운영체제(OS)가 파일에 대한 관리를 한다.
- 따라서 프로그램은 운영체제에 파일처리를 요청하여 결과를 받는다.



# 파일 입/출력

---

## 파일 입/출력 기초

- 프로그램이 파일 처리를 요청하는 방법
- 파일을 열어 놓는 것은 컴퓨터 자원을 소모
- 파일사용이 끝나면 반드시 파일을 닫는다.





# 파일 입/출력

---

## 파일 입/출력 기초

- 프로그램이 파일 처리를 요청하는 방법
- Python에서 사용하는 함수



# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

- 파일 객체명 = open(file, mode='r', encoding=None)
  - ※ open함수에는 더 많은 매개변수가 있으나, 일부만 소개한다.
- file
  - 실제 파일의 경로를 나타내는 문자열을 넘긴다.
  - 파일명만 입력할 경우 실행한 위치의 디렉터리에서 파일을 찾는다.

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

- file

- 파일명만 입력할 경우 실행한 위치의 디렉터리에서 파일을 찾는다.

- Path

- 프로그램(python, 운영체제 등)에서 파일이나 모듈(라이브러리)를 찾는 위치
- Python에서는 sys.path를 통해 path를 확인할 수 있다.

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

### —Path

- Python에서는 sys.path를 통해 path를 확인할 수 있다.
- .py를 실행할 경우 맨 앞에 실행한 파일의 현재 위치가 나온다.
- path에 있는 순서대로 파일이나 모듈을 검색하여 사용한다.
- 인터랙티브 모드(python3만 실행 시)에는 현재 경로를 받지 않는다.

```
1  import sys
2
3  print(sys.path)
```

```
root@goorm:/workspace/PythonSeminar18/TeachingMaterials/insert_file/23_lecture_
aster)# python3 path.py
['/workspace/PythonSeminar18/TeachingMaterials/insert_file/23_lecture_file', '/c
cal/lib/python36.zip', '/usr/local/lib/python3.6', '/usr/local/lib/python3.6/li
oad', '/usr/local/lib/python3.6/site-packages']
```

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

—파일 객체명 = open(file, mode='r', encoding=None)

—mode: 파일을 읽고 쓰는 모드이다.

문자	의미
'r'	읽기(기본값)
'w'	쓰기. 단, 파일이 존재 시 파일내용을 지움
'x'	베타적 생성모드로 열기. 파일 존재 시 IOError 예외 발생
'a'	쓰기. 단, 파일이 존재 시 기존 내용에 덧붙임
'b'	바이너리 모드
't'	텍스트 모드(기본값)
'+'	읽기/쓰기용으로 파일 열기

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

—파일 객체명 = open(file, mode='r', encoding=None)

—mode: 파일을 읽고 쓰는 모드이다.

문자	의미
'r+'	파일을 읽고 쓴다. 단, 파일이 없으면 만들지 않고, 있으면 내용을 지운다.
'w+'	파일을 읽고 쓴다. 단, 파일이 없으면 파일을 만들고, 있으면 내용을 지운다.
'a+'	파일을 읽고 쓴다. 단, 파일이 없으면 파일을 만들고, 있으면 내용에 덧붙인다.

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

- 파일 객체명 = `open(file, mode='r', encoding=None)`
- mode: 파일을 읽고 쓰는 모드이다.
  - 기본값은 r(읽기)과 t(텍스트 모드)이다.
  - mode = 'rt'나 'r', 't', mode 매개변수를 안 쓴 것은 똑같은 매개변수를 입력한 것으로 처리된다.

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

- 파일 객체명 = `open(file, mode='r', encoding=None)`
- encoding
  - 파일의 인코딩을 지정한다.
  - 파일을 읽을 때 필요한 중요한 옵션이다.
  - 파일의 비트열을 어떻게 읽을지 알려주는 옵션이다.



# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

- 파일 객체명 = `open(file, mode='r', encoding=None)`
- encoding
  - Linux계열에서는 문제가 없으나(기본적으로 UTF-8), 윈도우의 기본 인코딩은 cp949로 되어있다.
  - Python 3는 내부에서 UTF-8을 사용하기 때문에 윈도우에서 파일을 입출력할 때, 문제가 발생할 수 있다.

# 파일 입/출력-열기

---

## 파일 입/출력-열기(open)

—파일 객체명 = open(file, mode='r', encoding=None)

—encoding

— locale.getpreferredencoding()에 의해 기본 인코딩이 결정되나, 다르게 변경될 수 있으므로, 명시하는 것이 좋다.

— encoding = 'utf-8'

— 구름 IDE에서의 인코딩

```
>>> locale.getpreferredencoding()  
'UTF-8'
```

# 파일 입/출력-닫기

---

## 파일 입/출력-닫기(close)

- 파일 객체명.close()
- 파일을 명시적으로 닫아준다.

# 파일 입/출력-쓰기

---

## 파일 입/출력-쓰기(write)

- 파일객체명.write(출력문자열)를 입력하면 파일에 출력이 된다.
- write메서드는 파일에 쓴 바이트수를 반환한다.

# 파일 입/출력-쓰기

---

## 파일 입/출력-쓰기(write)

- 파일에 문자열을 기록하는 프로그램

```
1 file = open('test.txt', 'w', encoding='utf-8')
2 file.write('Hello World!')
3 file.close()
```

```
root@goorm:/workspace/PythonSeminar18/
re_file(master)# python3 write.py
root@goorm:/workspace/PythonSeminar18/
```



The screenshot shows a text editor window with two tabs: 'test...' and 'start...'. The 'test...' tab is active, and the text 'Hello World!' is written on line 1.

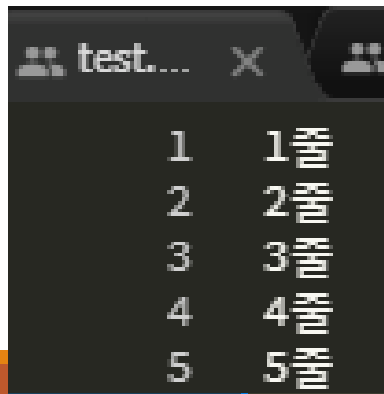
# 파일 입/출력-쓰기

---

## 파일 입/출력-쓰기(write)

- 파일에 여러 문자열을 기록하는 프로그램
- \n을 문자열에 넣어준다.

```
1 file = open('test.txt', 'w', encoding='utf-8')
2 for index in range(1, 6):
3     write_line = "%d줄\n" % index
4     file.write(write_line)
5 file.close()
```



1	1줄
2	2줄
3	3줄
4	4줄
5	5줄

# 파일 입/출력-쓰기

---

## 파일 입/출력-여러 줄 쓰기(writelines())

- 문자열을 요소로 가지는 리스트나 튜플 같은 시퀀스 자료형을 매개변수로 받아 이 객체의 내용을 모두 파일에 기록한다.

```
1 write_lines = [  
2     "Hello World!\n",  
3     "Python is Easy\n",  
4     "write text"  
5 ]  
6 file = open('multiline.txt', 'w', encoding='utf-8')  
7 file.writelines(write_lines)  
8 file.close()
```

```
1 Hello World!  
2 Python is Easy  
3 write text
```

# 파일 입/출력-읽기

---

## 파일 입출력-읽기(read(size))

- 파일 내용 전체를 읽어 문자열로 반환한다.
- size를 입력하면 size byte까지 읽어 반환한다.
- size가 없거나, 음수이면 파일 내용 전체를 읽는다.

```
1 file = open("test.txt", 'r', encoding='utf-8')
2 read_file = [file.read()]
3 file.close()
4 print(read_file)
```

```
root@goorm:/workspace/PythonSeminar_file(master)# python3 read.py
['1줄 \n2줄 \n3줄 \n4줄 \n5줄 \n']
```



# 파일 입/출력-읽기

---

## 파일 입출력-읽기(readline())

- 개행 문자(\n, \r, \r\n)를 기준으로 한 줄씩 읽는다.
- open()함수의 newline을 통해 개행문자를 변경할 수 있다.
- 문자열의 개행을 유지
- 마지막 문자열이면 빈문자열을 반환한다.

# 파일 입/출력-읽기

---

## 파일 입출력-읽기(readline())

—예시

```
1 file = open('test.txt', 'r', encoding='utf-8')
2 while True:
3     line = file.readline()
4     if not line:
5         break
6     print(line, end='')
7 file.close()
8
```

```
1 줄
2 줄
3 줄
4 줄
5 줄
```

# 파일 입/출력-읽기

---

## 파일 입출력-읽기(readlines())

- 파일 내용 전체를 읽어 개행 문자를 기준으로 요소를 가진 리스트를 반환한다.

```
1 file = open('test.txt', 'r', encoding='utf-8')
2 lines = file.readlines()
3 file.close()
4
5 print(lines)
6 for line in lines:
7     print(line, end='')
8
```

```
['1줄\n', '2줄\n', '3줄\n', '4줄\n', '5줄\n']
1줄
2줄
3줄
4줄
5줄
```

# 파일 입/출력-with문

---

## with~as문

- with open() as 파일객체명:  
수행문1  
수행문2
- with문을 빠져나가면 close가 자동으로 실행된다.
- Context manager에 의해 가능하다. -심화과정

# 파일 입/출력-with문

---

with~as문

—예시

```
1  with open('test.txt', 'r', encoding='utf-8') as file:
2      while True:
3          line = file.readline()
4          if not line:
5              break
6          print(line, end='')
7
8  file.tell()
9
```

1줄  
2줄  
3줄  
4줄  
5줄

```
Traceback (most recent call last):
  File "withas.py", line 8, in <module>
    file.tell()
ValueError: I/O operation on closed file.
```

# 명령행에서 입력 인수 받기

---

## 명령어의 옵션

—ex) `ls -a`

`ls`라는 명령어 뒤에 `-a`를 입력받는다.

—이렇게 입력 인수를 받도록 만들 수 있다.

# 명령행에서 입력 인수 받기

---

## c언어

- `int main(int argc, char* argv[])`로 시작하는 경우가 있다.
- `argc`(arguments count)는 `main`함수에 전달되는 정보의 개수
- `argv[]`(arguments vector)는 `main`함수에 전달되는 문자열의 배열이다.
- `argv[0]`은 프로그램의 실행경로

# 명령행에서 입력 인수 받기

---

## Python

- sys.argv를 통해 입력인수를 받는다.
- sys.argv[0]은 파일의 이름이다.
- Ex) sys\_input.parameter.py

```
1  import sys
2
3  for parameter in sys.argv:
4      print(parameter)
```

```
python3 sys_input_parameter.py aa bb cc adc
```

```
sys_input_parameter.py
aa
bb
cc
adc
```



# 예외 처리

---

## 예외

- 문법적으로 문제가 없으나, 실행하는 중 발생한다.
- 프로그램이 명령을 실행하는 도중 실패하는 것
- 보통 사용자가 잘못된 값을 입력하거나, 프로그래머의 실수로 만들어진다.
- Ex)

```
>>> a = int(input("정수를 입력하세요 "))
정수를 입력하세요 adfg
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'adfg'
```

# 예외 처리

---

## 예외처리

- 예외가 발생했을 때 처리하는 방법을 만든다.

# 예외 처리

---

try-except문

try:

# 실행시켜 문제가 없는지 확인할 코드

except [발생오류명 [as 오류 메시지 할당 변수]]:

# try 블록에서 문제가 생겼을 때 실행할 코드

# 예외 처리

---

## try-except문

- except만 사용
- 어떤 오류가 나든지 except블록을 실행
- pep8에서는 명시적으로 예외를 잡지 않기 때문에 좋은 예외 처리 방식이 아니라고 표시한다.
- 이는 실제 프로그래밍 에러가 나타나지 않게 하기 때문이다.

```
1 number_list = [1, 2, 3]
2
3 try:
4     while True:
5         index = int(input("인덱스를 입력하세요."))
6         print(number_list[index])
7
8 except:
9     print("오류 발생")
10
```

```
인덱스를 입력하세요 .2
3
인덱스를 입력하세요 .5
오류 발생
```

# 예외 처리

## try-except문

- except 발생오류명:
- 미리 정해놓은 발생오류명과 일치하면 except를 실행
- except (발생오류명1, 발생오류명2, ...): 으로 여러 오류를 묶어서 처리할 수 있다.

```
1 number_list = [1, 2, 3]
2
3 try:
4     while True:
5         index = int(input("인덱스를 입력하세요. "))
6         print(number_list[index])
7         print(number_list[index]/index)
8
9 except IndexError:
10     print("인덱스 범위가 아닙니다.")
11
12 except ZeroDivisionError:
13     print("0으로 나눌 수 없습니다.")
14
```

인덱스를 입력하세요 .1

2

2.0

인덱스를 입력하세요 .0

1

0으로 나눌 수 없습니다 .

인덱스를 입력하세요 .5

인덱스 범위가 아닙니다 .

# 예외 처리

---

## try-except문

- except 발생오류명 as 변수명(e나 err를 자주 사용):
- 발생오류명은 에러정보를 담고 있는 자료형이다.
- as의 뒤에 있는 변수명에 에러 인스턴스를 담는다.

# 예외 처리

---

## try-except문

- except 발생오류명 as 변수명(e나 err를 자주 사용):
- 예시

```
number_list = [1, 2, 3]

try:
    while True:
        index = int(input("인덱스를 입력하세요. "))
        print(number_list[index])
        print(number_list[index]/index)

except IndexError as e:
    print("인덱스 범위가 아닙니다.")
    print(type(e))
    print(e)

except ZeroDivisionError as e:
    print("0으로 나눌 수 없습니다.")
    print(type(e))
    print(e)
```

```
인덱스를 입력하세요 .5
인덱스 범위가 아닙니다 .
<class 'IndexError'>
list index out of range
```

```
인덱스를 입력하세요 .0
1
0으로 나눌 수 없습니다 .
<class 'ZeroDivisionError'>
division by zero
```

# 예외 처리

---

try-except문

- 더 많은 내장 에러에 대해서는 다음을 참고한다.
- <https://docs.python.org/ko/3/library/exceptions.html#builtin-exceptions>



# 예외 처리

---

## try-except-else문

- except가 실행되지 않으면 else문이 실행된다.
- if else와 유사한 구문으로 except에 대한 else이다.

# 예외 처리

---

## try-except-else문

—try:

# 실행시켜 문제가 없는지 확인할 코드

except [발생오류명 [as 오류 메세지 할당 변수]]:

# try 블록에서 문제가 생겼을 때 실행할 코드

else:

# except절이 실행되지 않으면 실행할 코드

# 예외 처리

---

try-except-else문

—예시

```
1  number_list = [1, 2, 3]
2
3  try:
4      index = int(input("인덱스를 입력하세요. "))
5      print(number_list[index])
6
7  except IndexError as e:
8      print("인덱스 범위가 아닙니다.", e)
9
10 else:
11     print("인덱스 범위입니다.")
12
```

```
인덱스를 입력하세요 .1
2
인덱스 범위입니다 .
```

```
인덱스를 입력하세요 .3
인덱스 범위가 아닙니다. list index out of range
```

# 예외 처리

---

## finally문

- 예외가 발생해도, 발생하지 않아도 무조건 실행
- 파일이나 통신같이 컴퓨터의 자원을 사용했을 때 이를 닫기 위해 사용

# 예외 처리

---

finally문

—예시

```
1  number_list = [1, 2, 3]
2
3  try:
4      index = int(input("인덱스를 입력하세요."))
5      print(number_list[index])
6
7  except IndexError as e:
8      print("인덱스 범위가 아닙니다.", e)
9
10 finally:
11     print("언제나 실행됩니다.")
```

```
인덱스를 입력하세요 .1
2
언제나 실행됩니다 .
```

```
인덱스를 입력하세요 .4
인덱스 범위가 아닙니다 . list index out of range
언제나 실행됩니다 .
```

# 예외 처리

## try-except-else-finally

—네 가지 문법을 다 합쳐서 사용할 수 있다.

```
1 number_list = [1, 2, 3]
2
3 try:
4     index = int(input("인덱스를 입력하세요. "))
5     print(number_list[index])
6
7 except IndexError as e:
8     print("인덱스 범위가 아닙니다.", e)
9
10 else:
11     print("인덱스 범위입니다.")
12
13 finally:
14     print("언제나 실행됩니다.")
15
```

```
인덱스를 입력하세요 .1
2
인덱스 범위입니다 .
언제나 실행됩니다 .
```

```
인덱스를 입력하세요 .5
인덱스 범위가 아닙니다 . list index out of range
언제나 실행됩니다 .
```

# 예외 처리

## 예외-예시

—입력된 숫자가 실수인지 판단

```
1  def is_number(num):
2      try:
3          float(num)
4      except ValueError:
5          return False
6      return True
7
8
9  num = input("부동소수점을 입력하세요: ")
10 while not is_number(num):
11     num = input("부동소수점을 입력해주세요: ")
12 num = float(num)
13 print(num)
```

```
부동소수점을 입력하세요: d
부동소수점을 입력해주세요:
부동소수점을 입력해주세요: 123
123.0
```

# 예외 처리

---

## 예외 처리과정

- 예외가 발생할 경우, 예외를 처리해줄 수 있는 상위 코드블록으로 올라간다.
- 계속 올라가도 찾을 수 없을 경우 코드 실행이 중지

```
1 def error_raise():
2     a = 2/0
3
4
5 def error_raise_call():
6     print("error raise call")
7     error_raise()
8
9
10 print("시작")
11 error_raise_call()
```

```
시작
error raise call
Traceback (most recent call last):
  File "except_process.py", line 11, in <module>
    error_raise_call()
  File "except_process.py", line 7, in error_raise_call
    error_raise()
  File "except_process.py", line 2, in error_raise
    a = 2/0
ZeroDivisionError: division by zero
```



# 예외 처리

---

## 예외 처리과정

- try-except가 있는 곳에서 예외가 처리된다.

```
1  def error_raise():
2      a = 2/0
3      print("예외로 인해 실행 안됨")
4
5
6  def error_raise_call():
7      print("error raise call")
8      try:
9          error_raise()
10     except ZeroDivisionError:
11         print("예외를 처리합니다.")
12
13
14  print("시작")
15  error_raise_call()
16  print("종료")
```

```
시작
error raise call
예외를 처리합니다.
종료
```

# 예외 처리

---

## 예외 발생시키기(raise)

- "0으로 나누기"같은 오류도 프로그래머가 예외를 발생시키기 때문에 나타난다.
- 작성한 코드에서 예외를 발생시킬 수 있다.
- `raise Exception(['에러메시지'])`
- ※ Exception 클래스를 통해 에러를 만든다.

# 예외 처리

---

## 예외 발생시키기(raise)

```
1 def check_value():
2     num = int(input("5의 배수를 입력하세요: "))
3     if num % 5 != 0:
4         raise Exception('5의 배수가 아닙니다!')
5     print(num)
6
7
8 def check_value_call():
9     check_value()
10
11
12 print("시작")
13 check_value()
14 print("종료")
```

시작

5의 배수를 입력하세요: 3

Traceback (most recent call last):

File "error\_raise.py", line 13, in <module>  
 check\_value()

File "error\_raise.py", line 4, in check\_value  
 raise Exception('5의 배수가 아닙니다!')

Exception: 5의 배수가 아닙니다!

# 예외 처리

## 예외 발생시키기(raise)

—마찬가지로 except로 해결할 수 있다.

```
1 def check_value():
2     num = int(input("5의 배수를 입력하세요: "))
3     if num % 5 != 0:
4         raise Exception('5의 배수가 아닙니다!')
5     print(num)
6
7
8 def check_value_call():
9     try:
10        check_value()
11    except Exception as err:
12        print(err)
13        print("예외를 처리합니다.")
14
15
16 print("시작")
17 check_value_call()
18 print("종료")
```

시작

5의 배수를 입력하세요: 23

5의 배수가 아닙니다!

예외를 처리합니다.

종료

# 예외 처리

---

## 예외 재발생시키기(raise)

- 예외를 상위 호출자에게 던질 수 있다.
- raise만 입력

# 예외 처리

## 예외 재발생시키기(raise)

```
1  def check_value():
2      num = int(input("5의 배수를 입력하세요: "))
3      if num % 5 != 0:
4          raise Exception('5의 배수가 아닙니다!')
5      print(num)
6
7
8  def check_value_call():
9      try:
10         check_value()
11     except Exception as err:
12         print(err)
13         print("예외를 상위 호출자에게 던집니다.")
14         raise
15
16
17 print("시작")
18 try:
19     check_value_call()
20 except Exception as err:
21     print(err)
22     print("예외를 처리합니다.")
23 print("종료")
```

시작

5의 배수를 입력하세요: 3

5의 배수가 아닙니다!

예외를 상위 호출자에게 던집니다.

5의 배수가 아닙니다!

예외를 처리합니다.

종료

# 예외 처리

---

## 예외 발생시키기(assert)

- assert 조건식[, 에러메시지]
- 지정된 조건식이 거짓일 때 AssertionError를 발생
- 프로그래머가 기본 조건을 바꾸었을 때, 나오면 안되는 조건이 나와 버그가 발생하는 것을 미리 확인하려고 사용한다.
- 디버깅 모드에서만 실행
- 파이썬은 기본적으로 디버깅모드
- assert를 실행하지 않으려면 python -O 파일.py로 실행

# 예외 처리

---

## 예외 발생시키기(assert)

—Ex) 그림 맞추기 게임을 만들 때 개수 설정

```
1  # 블록의 행과 열에 대한 상수
2  BLOCK_ROW = 5
3  BLOCK_COLUMN = 7
4
5  assert BLOCK_ROW * BLOCK_COLUMN % 2 == 0, '블록개수가 짝수가 아닙니다.'
```

```
Traceback (most recent call last):
  File "assert.py", line 5, in <module>
    assert BLOCK_ROW * BLOCK_COLUMN % 2 == 0, '블록개수가 짝수가 아닙니다.'
AssertionError: 블록개수가 짝수가 아닙니다.
```



# 예외 처리

---

## 사용자 예외

- Exception 클래스를 상속받아 만든다.
- Class에서 만들어 본다.

# 중간과제-숫자야구

---

## 숫자야구(number\_baseball.py)

- 1. 컴퓨터가 임의의 네 자리 수를 만든다.
- 2. 사용자가 임의의 네 자리 수를 입력한다.
  - 1) 임의의 숫자는 각 자리의 숫자가 모두 다른 숫자이다.  
즉, 0000-9999의 숫자이고, 앞자리의 0은 생략되지 않는다.  
Ex) 0123 (O) 0012(X)
  - 2) 네 자리 수가 아닌 다른 자리수의 수(ex, 다섯 자리, 세자리 등)을 입력할 경우 "입력이 잘못되었습니다."를 출력한 후 다시 입력을 받는다.

# 중간과제-숫자야구

---

## 숫자야구(number\_baseball.py)

- 3. 컴퓨터는 사용자가 입력한 네 자리 수를 확인하여 스트라이크(S), 볼(B), 아웃(O)의 개수를 알려준다.
  - 컴퓨터와 사용자가 입력한 수를 비교하였을 때,
    - 1) 스트라이크(S)는 컴퓨터가 가진 숫자와 비교해서 숫자와 숫자의 위치가 같은 개수이다
    - 2) 볼(B)은 컴퓨터가 가진 숫자와 비교해서 숫자가 같은 개수이다.
    - 3) 아웃(O)은 볼도 스트라이크도 아닌 숫자의 개수이다.
  - 단, 스트라이크일 경우는 볼로 계산하지 않는다.

# 중간과제-숫자야구

---

## 숫자야구(number\_baseball.py)

### —4. 맞췄을 경우

"축하합니다. x번만큼 질문하여 맞추셨습니다."  
를 출력하고

"다시 하시겠습니까?(yes/no)"를 출력한다.

— 1) yes(or y)를 입력할 경우 게임을 재시작한다.

— 2) no(or n)를 입력할 경우 게임을 종료한다.

— 3) 다른 문자열을 입력할 경우

"다시 하시겠습니까?(yes/no)"를 다시 출력한다.

# 중간과제-숫자야구

---

## 컴퓨터적 사고

- 문제가 주어질 때 문제정의를 되어있다.
- 필요한 활동을 생각해보자

# 중간과제-숫자야구

---

## 컴퓨터적 사고-분할정복

- 컴퓨터가 임의의 네 자리 수를 만든다.
- 사용자가 숫자를 입력한다.
- 사용자가 입력한 숫자와 컴퓨터의 숫자를 비교해 스트라이크, 볼, 아웃의 개수를 파악한다.
- 사용자가 입력한 숫자가 컴퓨터의 숫자와 같으면 종료한다.

# 중간과제-숫자야구

---

## 컴퓨터적 사고-분할정복

- 컴퓨터가 임의의 네 자리 수를 만든다.
  - 각 자리의 숫자가 다 다른지 확인한다.
- 사용자가 숫자를 입력한다.
  - 숫자가 네 자리인지 확인한다.
  - 각 자리의 숫자가 다 다른지 확인한다.

# 중간과제-숫자야구

---

## 컴퓨터적 사고-분할정복

- 사용자가 입력한 숫자와 컴퓨터의 숫자를 비교해 스트라이크, 볼, 아웃의 개수를 파악한다.
- 사용자 입력 숫자의 첫 번째 자리 숫자와 컴퓨터 첫 번째 자리 숫자와 비교하여 같은지 확인한다. 같으면 strike의 숫자를 하나 늘린다.
- 사용자 입력 숫자의 첫 번째 자리 숫자와 컴퓨터의 다른 자리 숫자와 비교하여 같은지 확인한다. 같으면 ball의 숫자를 하나 늘린다.
- 위에서 확인되지 않으면 out의 숫자를 하나 늘린다.
- 다음 숫자로 가서 반복한다.



# 중간과제-숫자야구

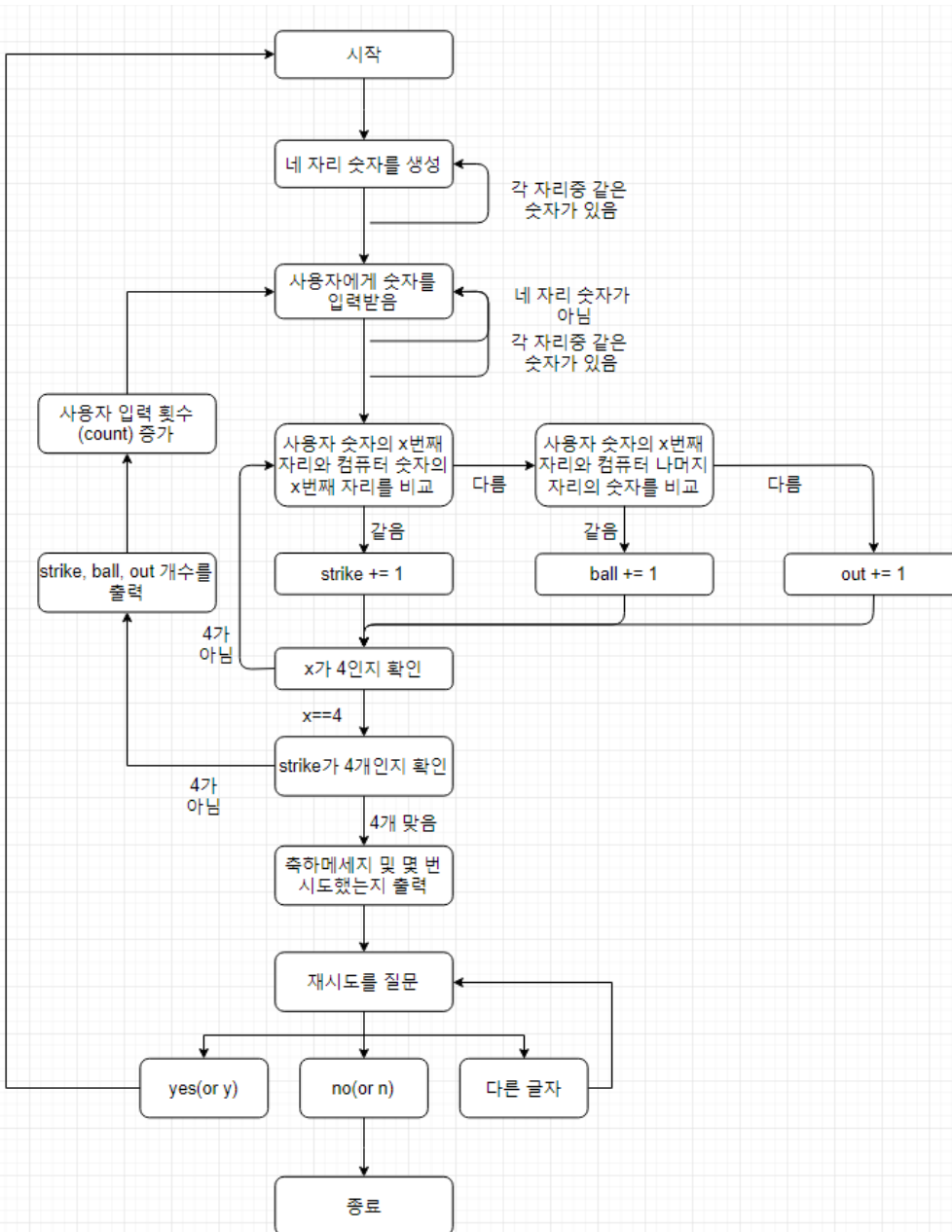
---

## 컴퓨터적 사고-분할정복

- 사용자가 입력한 숫자가 컴퓨터의 숫자와 같으면 종료한다.
- 몇 번 입력했는지 확인하여 출력한다.  
"축하합니다. x번 만큼 질문하여 맞추셨습니다."
- 다시 할 것인지 물어본다.
  - Yes(y)를 입력받으면 게임을 재시작한다.
  - no(n)를 입력받으면 게임을 종료한다.
  - 다른 값을 입력받으면 다시 물어본다.

# 중간과제-숫자 야구

## 컴퓨터적 사고-활동 다이어그램



# 중간과제-숫자야구

## 예시

```
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 1111
각 자리 숫자가 중복되지 않게 입력해주세요 .
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 15644
4자리 숫자를 입력해주세요 .
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 1234
strike: 0, ball: 2, out: 2
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 1256
strike: 0, ball: 1, out: 3
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 1356
strike: 0, ball: 1, out: 3
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 1456
strike: 0, ball: 2, out: 2
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4918
strike: 2, ball: 2, out: 0
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4891
strike: 2, ball: 2, out: 0
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 9841
strike: 0, ball: 4, out: 0
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4189
strike: 2, ball: 2, out: 0
```

# 중간과제-숫자야구

---

예시

```
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4819
strike: 1, ball: 3, out: 0
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4981
strike: 1, ball: 3, out: 0
중복되지 않은 4자리수를 입력해주세요 (0000-9999): 4198
strike: 4, ball: 0, out: 0
축하합니다. 10번만큼 질문하여 맞추셨습니다.
다시 하시겠습니까?(yes/no) gecs
다시 하시겠습니까?(yes/no) no
```

# 기본과제-자판기5

---

## 자판기(vending\_machine.py)

- 배운 내용을 바탕으로 새로운 기능을 추가한다.
- 이전의 프로그램에 추가하시오.
- 1. (파일이름).txt를 만들어 물품의 이름, 가격, 개수를 저장한다.
- 2. 프로그램을 실행할 때 (파일이름).txt를 불러와 프로그램에 로드한다.
- 3. 프로그램을 실행할 때 (파일이름).txt가 존재하지 않으면 "관리자에게 연락하십시오."를 출력하고 종료한다.

# 기본과제-자판기5

---

## 자판기(vending\_machine.py)

- 4. admin 모드에서 나갈 때, 변경 내역을 파일에 저장한다.
- 5. 돈을 반환할 때, 변경 내역을 파일에 저장한다.
  - ※ 잦은 파일 저장은 파일 저장 과정에서 병목이 일어나 프로그램이 느려질 수 있다
  - ※ 프로그램을 종료할 때, 돈이 반환되어 종료시 저장을 따로 지정하지 않는다.
- ※ 문자열을 원하는 delimiter(구분자)로 나누는 메서드는 문자열.split("구분자")를 사용한다. 강의파일 04번의 79페이지를 참조한다.

# 기본과제-자판기5

## 예시

```
-rwxrwxr--  1 root  root   82  1월 22 10:52 item_list.txt
-rwxrwxr--  1 root  root 7749  1월 23 09:26 24.vending_machine.py
```

관리자에게 연락하십시오.