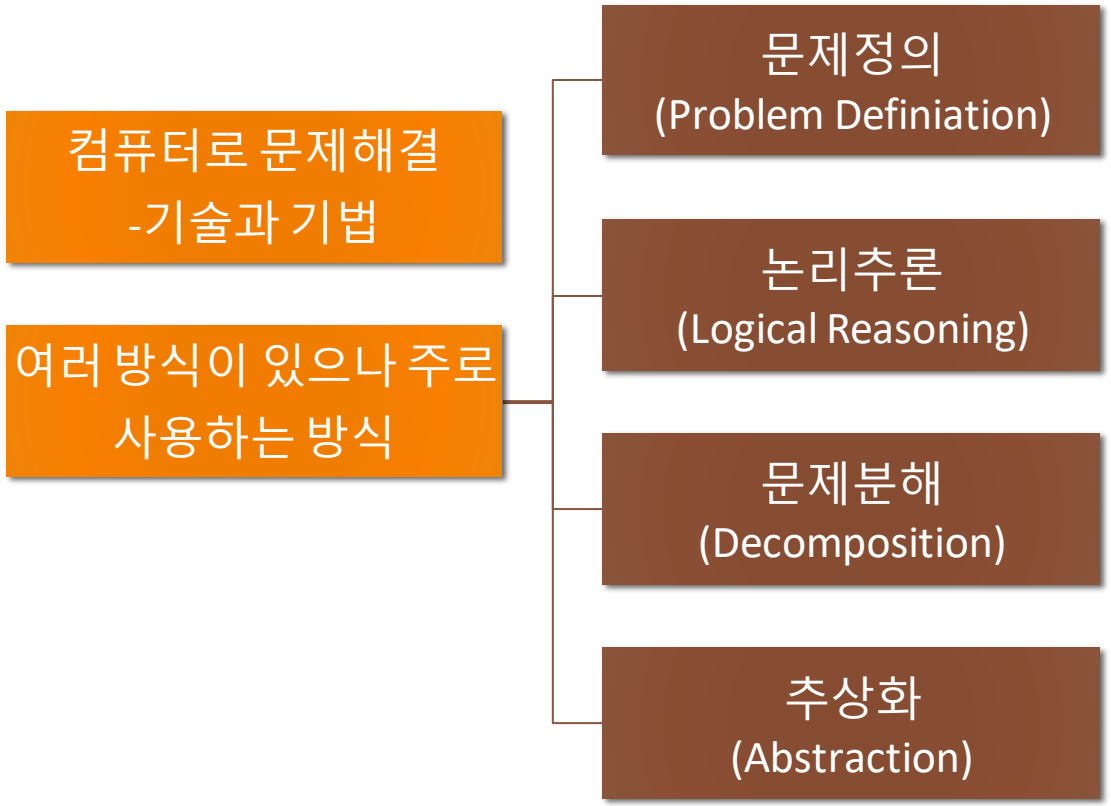
16. 문제분해, 자료형3₩ (딕셔너리, 집합), ₩ 반복문, import

2018.12

일병 김재형

문제해결


컴퓨터로 문제해결
-기술과 기법
여러 방식이 있으나 주로 사용하는 방식
문제정의
(Problem Definiation)
논리추론
(Logical Reasoning)
문제분해
(Decomposition)
추상화
(Abstraction)

문제해결3-문제분해

분할정복(divide and conquer)라고도 불린다.

- 하나의 문제를 여러 부분의 문제들로 나누고, 그 부분 문제들을 해결하여 문제를 해결

- 두 가지 주요 방법론
  - 하향식 설계
  - 연속 프로토타이핑

문제해결3-문제분해

예시

- 배가 고픈 상태에서 강 반대쪽 나무에 열매가 있는 것을 발견했다.

문제해결3-문제분해

예시

- 배가 고픈 상태에서 강 반대쪽 나무에 열매가 있는 것을 발견했다.
  - 1. 강을 건넌다.
  - 2. 나무 위에 있는 열매를 떨군다.
  - 3. 먹는다.

문제해결3-문제분해

예시

- 배가 고픈 상태에서 강 반대쪽 나무에 열매가 있는 것을 발견했다.
  - 1. 강을 건넌다.
    - 다리를 찾는다.
    - 뗏목을 만든다.
  - 2. 나무 위에 있는 열매를 떨군다.
  - 3. 먹는다.

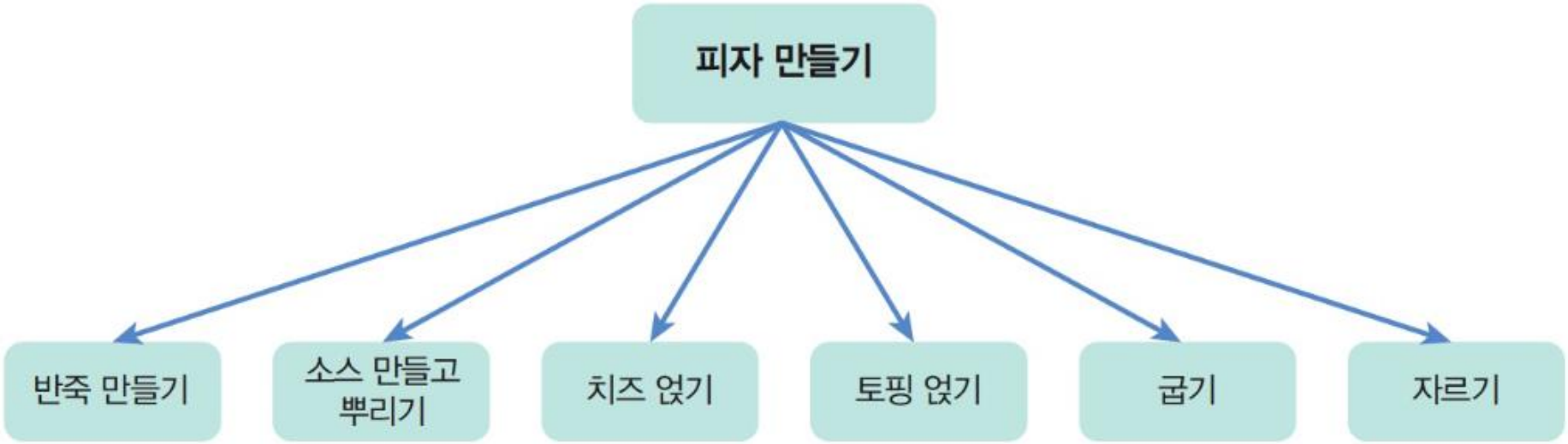
문제해결3-문제분해

예시

- 배가 고픈 상태에서 강 반대쪽 나무에 열매가 있는 것을 발견했다.
  - 1. 강을 건넌다.
  - 2. 나무 위에 있는 열매를 떨군다.
    - 나무에 올라간다.
    - 나무막대기로 친다.
    - 나무를 발로 찬다.
  - 3. 먹는다.

문제해결3-문제분해

예시



문제분해-하향식 설계

하향식설계

- 문제의 요약에서 시작한다.
  - 개요를 작성한다.
- 분명하지 않은 부분들을 다듬어 점차 구체적으로 분명하게 만든다.

문제분해-하향식 설계

하향식설계

- 설계(design) - 문제정의 단계에서 정리된 소프트웨어 요구사항 명세서에 기술된 내용을 실제로 구현할 수 있도록하는 모든 개발과정

- 문제의 요약을 통해 분명하지 않은 명령어들을 명확하게 다듬어 표현

문제분해-하향식 설계

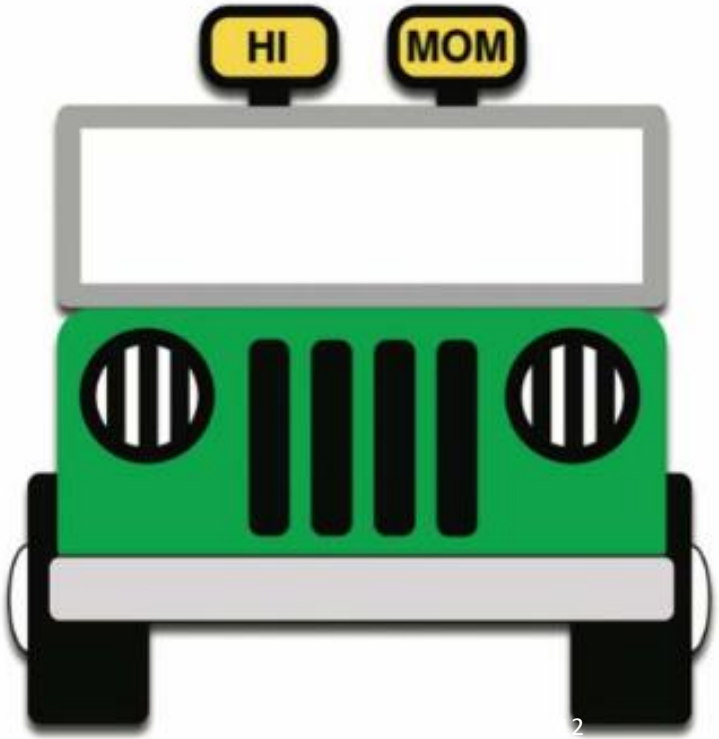
하양식설계

- 문제의 요약에서 시작한다.
  - 개요를 작성한다.
- 분명하지 않은 부분들을 다듬어 점차 구체적으로 분명하게 만든다.

문제분해-하향식 설계

하향식설계 예시-비포장 도로용 차량 그리기

- 1. 녹색 그릴을 그린다.
- 2. 그릴 바로 밑에 범퍼를 그린다.
- 3. 그릴과 범퍼 아래에 타이어를 그린다
- 4. 그릴 바로 위에 앞유리를 그린다.
- 5. 앞유리의 위에 보조등을 두 개 그린다.



문제분해-하향식 설계

하향식설계 예시-비포장 도로용 차량 그리기

- 1. 녹색 그릴을 그린다.
  - 1) 녹색 그릴의 배경을 그린다.
  - 2) 좌측 헤드라이트를 그린다.
  - 3) 그릴 앞면에 4개의 동일한 모양과 크기의 검정색 직사각형을 그린다.
  - 4) 우측 헤드라이트를 그린다.

문제분해-하향식 설계

하향식설계 예시-비포장 도로용 차량 그리기

- 1. 녹색 그릴을 그린다.
  - 1) 녹색 그릴의 배경을 그린다.
  - 2) 좌측 헤드라이트를 그린다.
    - (1) 좌측 헤드라이트의 테두리로서 검은 색 점선을 그린다
    - (2) 검은 색 테두리 안에 흰색점을 가운데 그린다.
    - (3) 헤드라이트롱 보호개용으로 세 개의 같은 모양의 직사각형을 그린다.

문제분해-연속 프로토타이핑

프로토타입(prototype)

- 어떤 물체의 대략적인 모습
- 최종 제품을 만들기 전에, 일부분만을 보여주어 최종 제품의 모습을 대략 예상할 수 있도록 함
- 예시) 아파트 모델 하우스

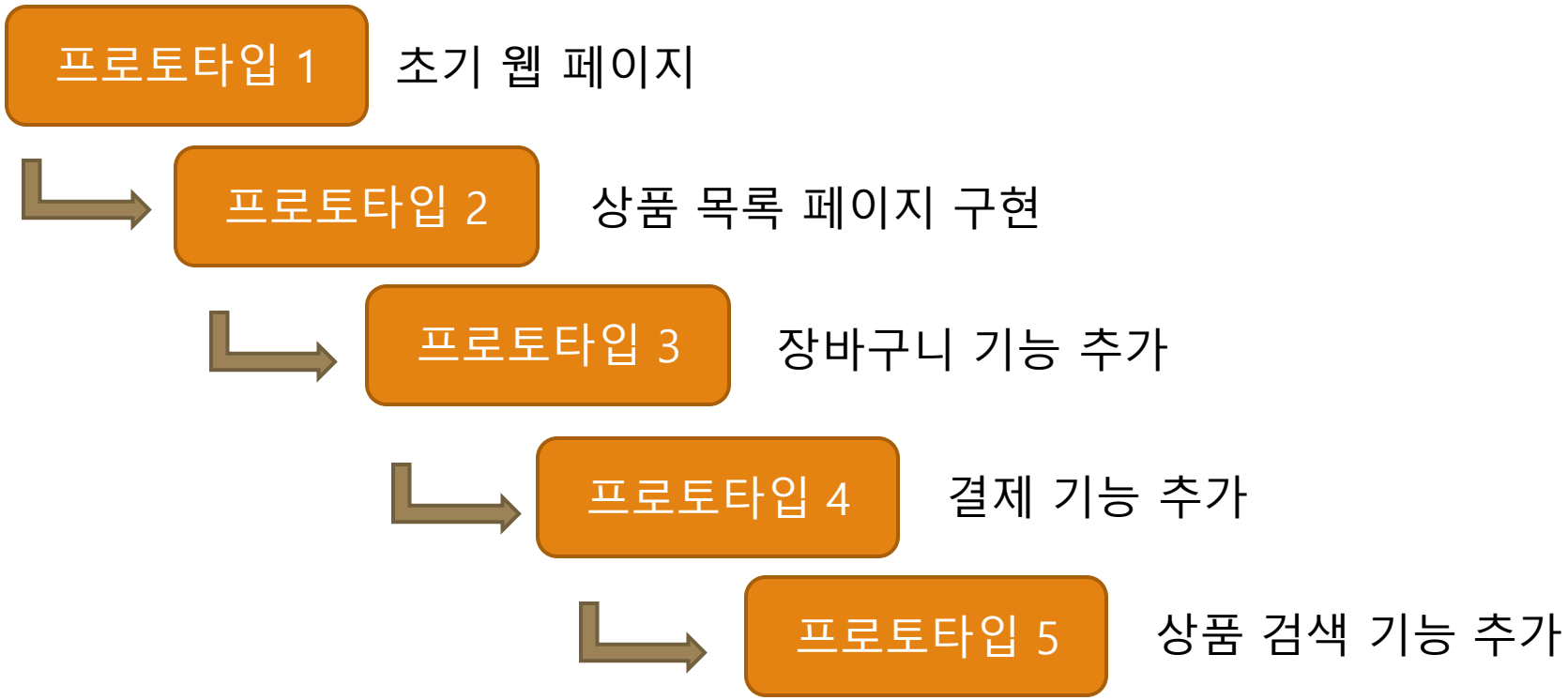
문제분해-연속 프로토타이핑

연속 프로토타이핑(successive prototyping)

- 프로토타입을 설계 단계별로 계속 만들어 설계 단계마다 제품의 최종 모습을 미리 어느 정도 파악할 수 있도록 함
- 각 단계에서 완성된 프로토타입은 고객에게 보여지고 피드백 받음

문제분해-연속 프로토타이핑

연속 프로토타이핑-웹페이지 만들기



문제분해-연속 프로토타이핑

자료형-딕셔너리

dictionary(사전, {})

- 리스트: 정수와 값을 대응
- Dictionary: 값(key)과 값(value)을 대응
- 즉, 처음부터 확인하는 것이 아니라 사전처럼 바로 단어를 찾아내는 것

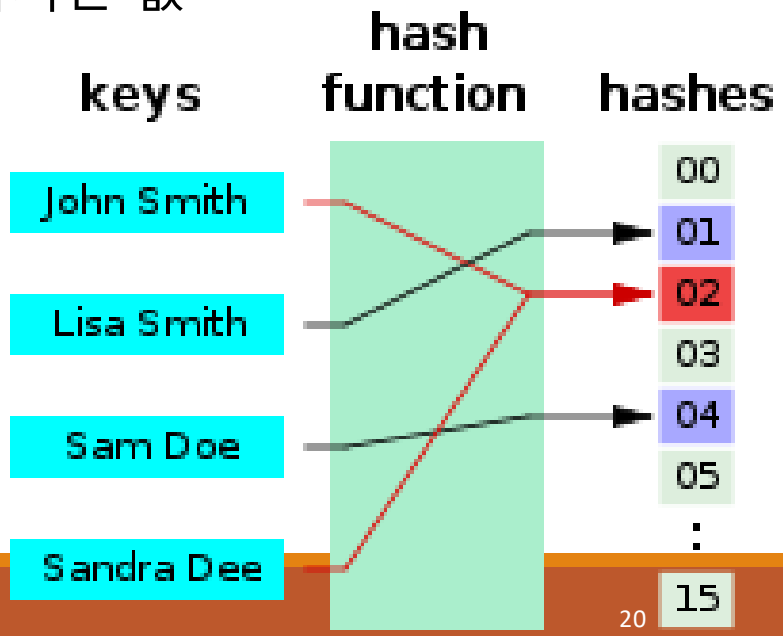
인덱스	값
0	"kim"
1	31
2	80

키	값
"name"	"kim"
"age"	31
"score"	80

자료형-딕셔너리

해시 테이블(hash table)

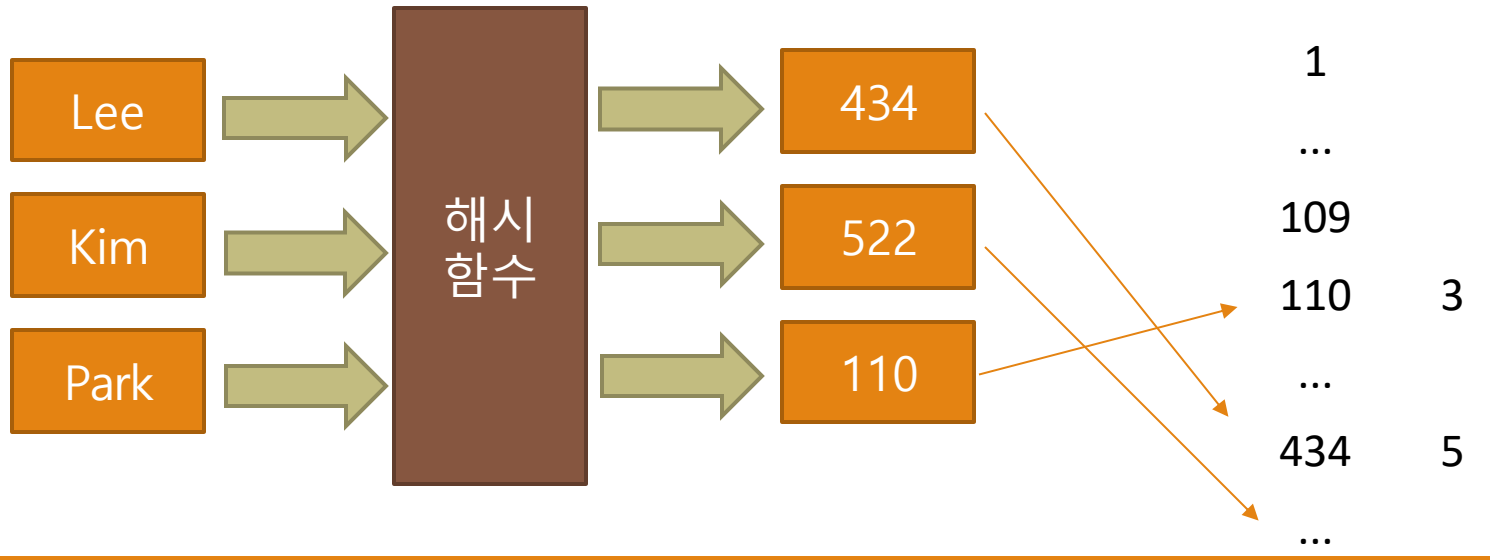
- 해시함수
  - 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수
  - 해시 값: 해시 함수에 의해 얻어지는 값



자료형-딕셔너리

해시 테이블(hash table)

- Key를 인수로 받아서 정수를 반환하는 해시 함수 이용
- 이를 통해 대응 관계를 표현한다.



자료형-딕셔너리

장점

- (값을 넣으면 바로 나옴으로) 속도가 빠르다.

단점

- 메모리 소비량이 크다
- 해시값의 충돌이 많으면 느리다.
- Key에는 불변형 자료형만 사용가능하다.
- 정렬이 불가능하다.

자료형-딕셔너리 생성

딕셔너리의 생성

- {Key1:Value1, Key2:Value2, Key3:Value3 ...}
- 단, Key에는 불변형 자료형만 가능

```
>>> names = {'name':"Jaehyeong", 'phone': '031-000-0000', 'birth': '6/29'}
>>> names
{'name': 'Jaehyeong', 'phone': '031-000-0000', 'birth': '6/29'}
>>> type(names)
<class 'dict'>
```

```
>>> a = {1:'hi'}
>>> b = {'a':[1, 2, 3]}
>>> a
{1: 'hi'}
>>> b
{'a': [1, 2, 3]}
```

자료형-딕셔너리 생성

딕셔너리의 중복키 생성

- 중복 시 맨 뒤의 값만 들어간다.

```
>>> names = {'name':"Jeahyeong", 'name':"Yeongchun"}
>>> names
{'name': 'Yeongchun'}
```

자료형-딕셔너리 생성

빈 딕셔너리와 dict()

- 빈 딕셔너리
  - a = {}
  - a = dict()

```
>>> a = {}
>>> a
{}
>>> a = dict()
>>> a
{}
```

- dict()으로 딕셔너리 만들기
  - a = dict(key1=value1, key2=value2)
  - b = dict([(key1, value1), (key2, value2)])
  - c = dict({key1: value1, key2: value2})

자료형-딕셔너리 쌍 추가/삭제

딕셔너리 쌍 추가하기

- 딕셔너리명[key] = value

```
>>> Jaehyeong = {}
>>> Jaehyeong
{}
>>> Jaehyeong['hp'] = 100
>>> Jaehyeong
{'hp': 100}
```

자료형-딕셔너리 쌍 추가/삭제

딕셔너리 쌍 삭제하기

- del 딕셔너리명[key]

```
>>> Jaehyeong
{'hp': 100}
>>> del Jaehyeong['hp']
>>> Jaehyeong
{}
```

자료형-딕셔너리 사용

딕셔너리 키에 접근

- 딕셔너리[key]를 입력하면 value가 나온다.

```
>>> Jaehyeong = {'hp': 100, 'mp': 400, 'armor': 12}
>>> Jaehyeong['hp']
100
```

- 없는 키를 넣으면 오류가 뜬다.

```
>>> Jaehyeong['melee']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'melee'
```

자료형-딕셔너리 사용

딕셔너리 키 확인

- 키 in 딕셔너리명
  - 키가 딕셔너리에 있으면 True

```
>>> 'hp' in Jaehyeong
True
```

- 키 not in 딕셔너리명
  - 키가 딕셔너리에 없으면 True

```
>>> 'hp' not in Jaehyeong
False
```

자료형-딕셔너리 사용

딕셔너리 키 개수 구하기

- len(딕셔너리명)

```
>>> Jaehyeong = {'hp': 100, 'mp': 250, 'armor': 12}
>>> len(Jaehyeong)
3
```

자료형-딕셔너리의 중첩

리스트와 같이 딕셔너리도 중첩이 된다.

- 딕셔너리 = {key1: {keyA: valueA}, key2: {keyB: valueB}}

```
terrestrial_planet = {
    'Mercury': {
        'mean_radius': 2439.7,
        'mass': 3.3022E+23,
        'orbital_period': 87.969
    },
    'Venus': {
        'mean_radius': 6051.8,
        'mass': 4.8676E+24,
        'orbital_period': 224.70069,
    }
}

print(terrestrial_planet['Mercury']['mass'])
```

```
3.3022e+23
```

중첩 딕셔너리의 복사

리스트와 같다.

- 단순 복제(a=b): 변수명만 다르다.
- 얕은 복사(a=b.copy()): 가장 겉의 딕셔너리만 복사
- 깊은 복사(a=b.deepcopy()): 내부까지 전체 복사

자료형-딕셔너리 메서드

딕셔너리 쌍 값 추가(setdefault(key))

- 딕셔너리에 키-쌍 값을 추가.
- 키만 지정하면 None 객체를 저장

```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> a.setdefault('d')
>>> a
{'a': 1, 'b': 2, 'c': 3, 'd': None}
```

자료형-딕셔너리 메서드

딕셔너리 쌍 값 추가(setdefault(키, 기본값))

- 키: 기본값으로 넣는다.

```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> a.setdefault('d', 100)
100
>>> a
{'a': 1, 'b': 2, 'c': 3, 'd': 100}
```

- 이미 있을 경우 추가되지 않는다.

```
>>> a.setdefault('d', 20)
100
>>> a
{'a': 1, 'b': 2, 'c': 3, 'd': 100}
```

자료형-딕셔너리 메서드

딕셔너리 키 값 수정(update(키=값))

- 딕셔너리에서 키의 값 수정
- ※ 키에는 문자열만 들어간다.

```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> a.update(a=100)
>>> a
{'a': 100, 'b': 2, 'c': 3}
```

자료형-딕셔너리 메서드

딕셔너리 키 값 수정(update(키=값))

- 키가 없으면 키-값 쌍을 추가한다.

```
>>> a
{'a': 100, 'b': 2, 'c': 3}
>>> a.update(b=20, d=15)
>>> a
{'a': 100, 'b': 20, 'c': 3, 'd': 15}
```

자료형-딕셔너리 메서드

딕셔너리 키 값 수정(update(키=값))

- 키가 숫자일 경우 update(딕셔너리)로 수정가능

```
>>> b = {1: 'a', 2: 'b'}
>>> b
{1: 'a', 2: 'b'}
>>> b.update({2: 'ab', 3: 'c'})
>>> b
{1: 'a', 2: 'ab', 3: 'c'}
```

자료형-딕셔너리 메서드

딕셔너리 키 값 수정(update(키=값))

- 키-값 쌍 여러 개를 콤마로 구분해서 넣어주면 한꺼번에 수정가능

```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> a.update(c=12, d=20, e=30)
>>> a
{'a': 1, 'b': 2, 'c': 12, 'd': 20, 'e': 30}
```

자료형-딕셔너리 메서드

딕셔너리 키 값 수정(update(키=값))

- update(리스트/튜플)
  단, 키-값 쌍으로 있어야 한다.
  [[키1, 값1], [키2, 값2]]/((키1, 값1), (키2, 값2))

```
>>> b
{1: 'a', 2: 'ab', 3: 'c'}
>>> b.update([[1, 'ac'], [4, 'd']])
>>> b
{1: 'ac', 2: 'ab', 3: 'c', 4: 'd'}
```

자료형-딕셔너리 메서드

딕셔너리 키의 값을 가져오기(get(키))

```
>>> a
{'a': 1, 'b': 2, 'c': 3}
>>> a.get('a')
1
>>> a.get('e')
>>> d=a.get('e')
>>> type(d)
<class 'NoneType'>
```

- Dic['nokey']는 에러 반환
- Dic.get('nokey')는 None반환

자료형-딕셔너리 메서드

딕셔너리 키 값 쌍 삭제(pop(키))

- 딕셔너리에서 특정 쌍을 삭제한 뒤 삭제한 값 반환

```
>>> a
{'a': 1, 'b': 2, 'c': 12, 'd': 20, 'e': 30}
>>> a.pop('d')
20
```

- pop(키, 기본값)
- 키가 없으면 기본값을 리턴

```
>>> a
{'a': 1, 'b': 2, 'c': 12, 'e': 30}
>>> a.pop('z', 10)
10
>>> a
{'a': 1, 'b': 2, 'c': 12, 'e': 30}
```

자료형-딕셔너리 메서드

딕셔너리의 모든 쌍을 삭제(clear())

- 모든 키-값 쌍을 삭제한다.

```
>>> a
{'a': 1, 'b': 2, 'c': 12, 'e': 30}
>>> a.clear()
>>> a
{}
```

자료형-딕셔너리 메서드

딕셔너리에서 키, 값, 키와 값 모두 가져오기

- items: 키-값 쌍을 모두 가져옴(dict_items객체)

```
>>> a
{'a': 1, 'b': 2, 'c': 3}
>>> b = a.items()
>>> b
dict_items([('a', 1), ('b', 2), ('c', 3)])
>>> type(b)
<class 'dict_items'>
```

자료형-딕셔너리 메서드

딕셔너리에서 키, 값, 키와 값 모두 가져오기

- keys: 키를 모두 가져옴(dict_keys객체)

```
>>> a
{'a': 1, 'b': 2, 'c': 3}
>>> b = a.keys()
>>> b
dict_keys(['a', 'b', 'c'])
>>> type(b)
<class 'dict_keys'>
```

자료형-딕셔너리 메서드

딕셔너리에서 키, 값, 키와 값 모두 가져오기

- values: 값을 모두 가져옴(dict_values객체)

```
>>> a
{'a': 1, 'b': 2, 'c': 3}
>>> b = a.values()
>>> b
dict_values([1, 2, 3])
>>> type(b)
<class 'dict_values'>
```

자료형-딕셔너리 메서드

dict_keys, dict_values, dict_items

- python 2.7에서는 리스트를 반환하여 메모리의 낭비가 있었다.
- python 3.x에서는 메모리 낭비를 방지하기 위해 위의 객체를 반환하도록 변경되었다.
- 반복 가능한 객체임으로 순환문이나 반복문에서 리스트와 같은 방법으로 사용하면 된다.

자료형-딕셔너리 메서드

리스트와 튜플로 딕셔너리 만들기

- dict.fromkeys(키 리스트/튜플)
  키 리스트로 딕셔너리를 생성, 값은 모두 None

```
>>> b = ['a', 'b', 'c']
>>> a = dict.fromkeys(b)
>>> a
{'a': None, 'b': None, 'c': None}
```

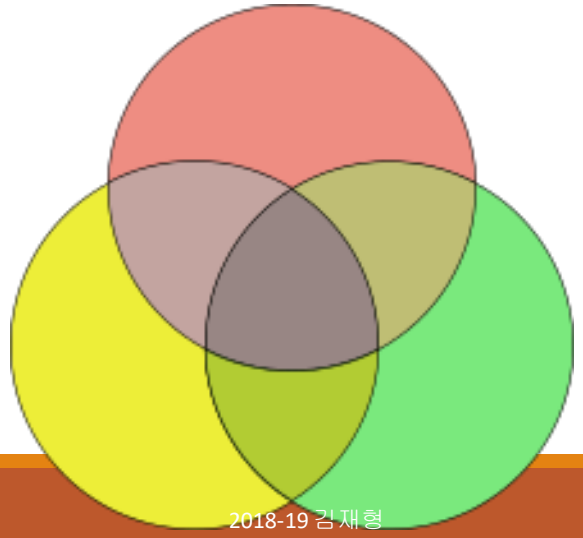
- dict.fromkeys(키 리스트/튜플, 기본값)
  키 리스트로 딕셔너리를 생성, 값은 모두 기본값

```
>>> b = ('a', 'b', 'c')
>>> a = dict.fromkeys(b, 10)
>>> a
{'a': 10, 'b': 10, 'c': 10}
```

자료형-집합

집합이란?

- 특정 조건에 맞는 원소(구성원, 대상)들의 모임
- *a*는 집합 *A*의 원소이다.'는 $a \in A$로 표시
- 벤 다이어그램: 집합을 원이나 타원등의 단일폐곡선으로, 원소는 점으로 나타내 집합간의 간단한 관계를 표현



자료형-집합

집합의 생성

- set()을 사용해 만든다.

```
>>> a = set([1, 2, 3, 4])
>>> a
{1, 2, 3, 4}
```

```
>>> b = set("Hello")
>>> b
{'o', 'l', 'H', 'e'}
```

자료형-집합

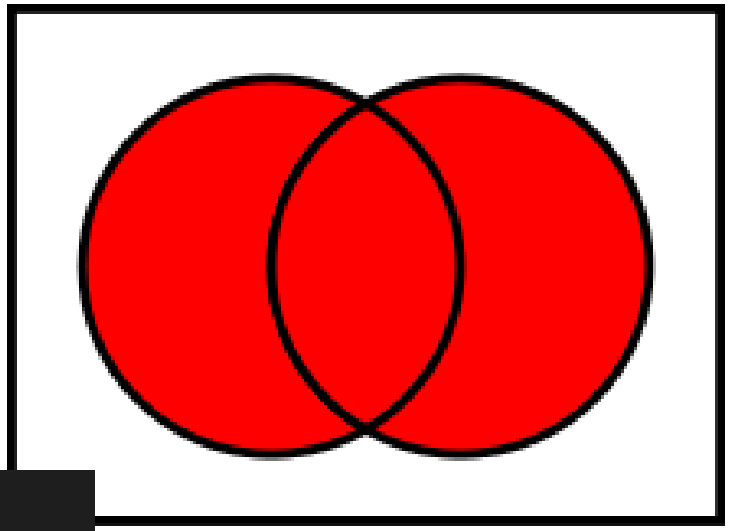
집합 자료형의 특징

- 중복을 허용하지 않는다.
  - 이를 통해 리스트나 튜플의 중복을 삭제할 수 있다.
- 순서가 없다(Unordered).
  - 인덱싱으로 접근할 수 없다.
  - 인덱싱 접근시 리스트나, 튜플로 변환한다

자료형-집합

합집합

- 여러 집합의 원소를 모두 모은 집합(|)
- union(집합)을 사용해도 된다.

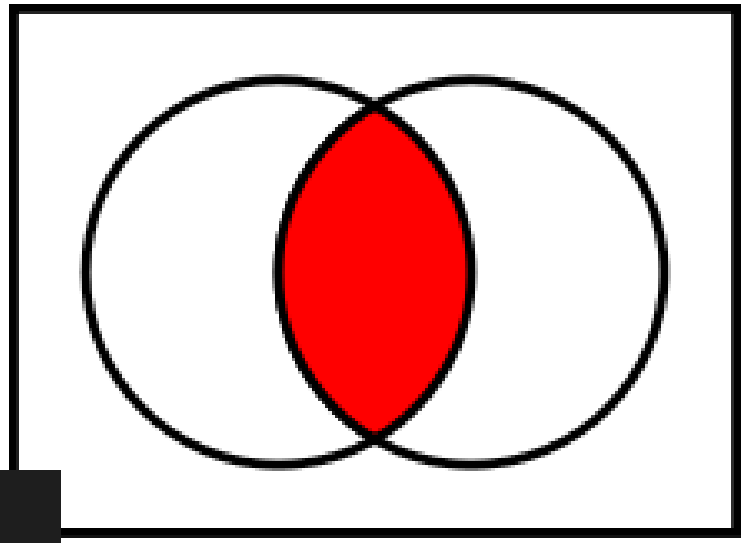


```
>>> s1 = set([1, 2, 3, 4, 5, 6, 7])
>>> s2 = set([6, 7, 8, 9, 10])
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

자료형-집합

교집합

- 여러 집합의 공통 원소를 모은 집합 (&)
- intersection(집합)을 사용해도 된다.

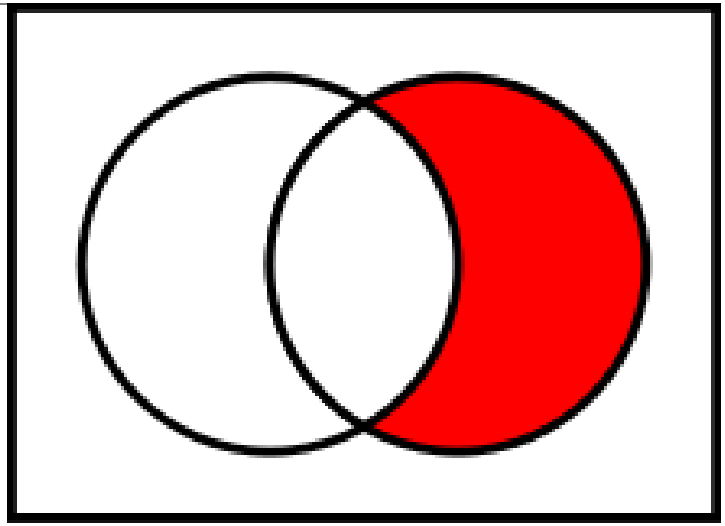


```
>>> s1 = set([1, 2, 3, 4, 5, 6, 7])
>>> s2 = set([6, 7, 8, 9, 10])
>>> s1 & s2
{6, 7}
>>> s1.intersection(s2)
{6, 7}
```

자료형-차집합

차집합

- 두 집합 사이의 겹치는 원소를 제외하는 연산 (-)
- differences(집합)을 사용할 수 있다.



```
>>> s1 = set([1, 2, 3, 4, 5, 6, 7])
>>> s2 = set([6, 7, 8, 9, 10])
>>> s1 - s2
{1, 2, 3, 4, 5}
>>> s2 - s1
{8, 9, 10}
```

```
>>> s1.difference(s2)
{1, 2, 3, 4, 5}
>>> s2.difference(s1)
{8, 9, 10}
```

자료형-집합 메서드

값 한 개 추가하기(add())

```
>>> s1
{1, 2, 3, 4, 5, 6, 7}
>>> s1.add(8)
>>> s1
{1, 2, 3, 4, 5, 6, 7, 8}
```

자료형-집합 메서드

값 여러 개 추가하기(update())

```
>>> s2
{6, 7, 8, 9, 10}
>>> s2.update([1, 2, 3, 4])
>>> s2
{1, 2, 3, 4, 6, 7, 8, 9, 10}
```

자료형-집합 메서드

값 제거하기(remove())

```
>>> s1
{1, 2, 3, 4, 5, 6, 7, 8}
>>> s1.remove(1)
>>> s1
{2, 3, 4, 5, 6, 7, 8}
```

반복문(while)

반복문

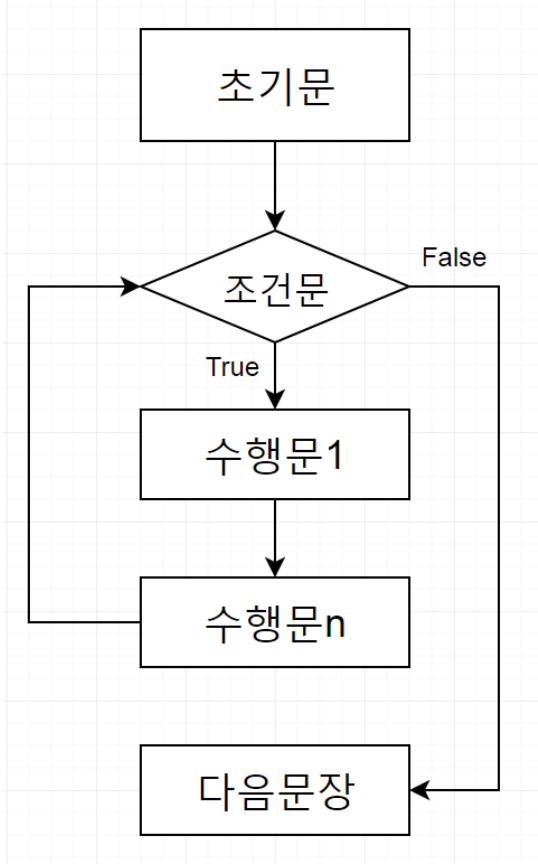
- 조건을 만족하고 있는 동안 블록 안의 내용을 반복하여 실행하는 구문
- while: ~하는 동안에

반복문(while)

반복문을 사용하지 않고 반복?

- if문을 goto문으로 사용하는 것과 같다.

```
1  void not_use_while(int x){
2      print("not_use_while\n");
3      LOOP:
4          if(x < 0) goto END;
5          print("loop!\n);
6      goto LOOP;
7      END:
8  }
```



반복문(while)

반복문도 goto문만 있으면 가능한 것

새로운 것이 아니라 읽기 쉽게, 쓰기 쉽게 한다.

반복문(while)

반복문의 구조

초기문

while 조건문:

    수행문1

    수행문2

    수행문n

    변화식

반복문(while)

반복문의 구조

- 초기문: 반복횟수를 제한하는 변수 정의
- 조건문: 반복횟수에 대한 제한 조건을 주는 식
- 변화식: 반복횟수를 제한하는 변수의 값을 변경

※ 변화식을 생략하면 무한히 반복된다(무한 루프)

반복문(while) 예시

```
limit = int(input("반복횟수: "))

count = 1
while count <= limit:
    print("%s회 반복 중" % count)
    count = count + 1
print("반복 끝!")
```

초기문

조건문

변화식

```
python while_example.py
반복횟수: 4
1회 반복 중
2회 반복 중
3회 반복 중
4회 반복 중
반복 끝!
```

pass, break, continue

pass(구현을 잠시 미뤄두기)

- 프로그래밍을 하다보면 그 구역을 나중에 구현하고 싶을 때가 있다.
- C언어는 {}(중괄호)를 사용해서 그냥 비운다.
- Python에서 그냥 비웠을 때는 코드블록으로 인정받지 않아 오류가 난다.

```
>>> count = 0
>>> if count >= 10:
...
...
  File "<stdin>", line 3

    ^
IndentationError: expected an indented block
```

pass, break, continue

pass(구현을 잠시 미뤄두기)

- pass 키워드를 사용하면 잠시 미뤄둘 수 있다.

```
>>> count = 0
>>> if count >= 10:
...     pass
...
>>>
```

pass, break, continue

break(반복문을 종료)

- 반복문을 수행하는 동안 반복문 자체를 빠져나가고 싶을 때 사용.
- break문을 만나면 가장 가까운 반복문을 빠져나간다.

```
>>> count = 0
>>> while count <= 10:
...     print("%s, count값" %count)
...     if count == 5:
...             print("종료")
...             break
...     count += 1
...
0, count값
1, count값
2, count값
3, count값
4, count값
5, count값
종료
```

pass, break, continue

continue(조건에 맞지 않으면 처음으로 가기)

- 조건에 맞지 않으면 나머지를 실행하지 않지만 반복은 계속 진행할 때 사용
- ex) 1부터 16까지 짝수만 출력

```
>>> count = 0
>>> while count <= 15:
...     count += 1
...     if count % 2 == 1:
...         continue
...     print(count)
...
2
4
6
8
10
12
14
16
```

반복문(while)

무한루프(무한반복)

- 반복문의 조건이 무조건 참이 되어 무한히 반복되는 것이다.
- 일반적인 프로그램에서 무한루프를 사용하지 않는 경우가 없다.
- 만일 무한루프에 걸렸을 경우 'Ctrl+C'로 빠져나온다. (KeyboardInterrupt)

※ 정 안되면 창을 끄고 다시 접속합시다.

반복문(while)

무한루프(무한반복)

```
1  while True:
2      answer = input("계속 반복합니까?(yes/no): ")
3      if answer == "no":
4          print("종료!")
5          break
6      elif answer == "yes":
7          print("계속 반복!")
8      else:
9          print("yes/no를 입력하세요.")
```

```
계속 반복합니까?(yes/no): yes
계속 반복!
계속 반복합니까?(yes/no): yes
계속 반복!
계속 반복합니까?(yes/no): eg
yes/no를 입력하세요.
계속 반복합니까?(yes/no): no
종료!
```

산술 할당 연산자

산술 연산을 수행 후 할당하는 연산자이다.

- a= a+b
  => a += b
- 간단하게 나타낼 때 사용

연산	원래 모양	축약
덧셈	a = a+b	a += b
뺄셈	a= a-b	a -= b
곱셈	a = a*b	a *= b
나눗셈	a = a/b	a /= b
몫	a = a//b	a //= b
나머지	a = a%b	a %= b

```
>>> a = 2
>>> b = 3
>>> a += b
>>> a
5
>>> a += 1
>>>
>>> a
6
```

내장라이브러리 사용하기

import

- 모듈은 함수나 변수, 클래스들을 모은 파일
- import는 다른 모듈내의 코드에 접근하도록 한다.
- 자세한 내용은 후에 설명한다.

내장라이브러리 사용하기

import

- import 모듈
  (모듈은 .py로 만드나, 확장자는 없이 사용한다.)
- 모듈이름.변수/함수이름으로 사용

ex) random 모듈 import

```
>>> import random
>>> random.randint(1, 100)
77
```

내장라이브러리 사용하기

import

- from 모듈 import 변수나 함수
  - 모듈 이름 없이 변수나 함수를 사용할 때 사용
- Ex) random모듈에서 randint함수 가져오기

```
>>> from random import randint
>>> randint(1, 100)
29
```

내장라이브러리 사용하기

import
- from 모듈 import 변수나 함수
  - 여러 줄을 쓰거나 ','로 한번에 쓸 수도 있다.
- Ex) random모듈에서 randint함수 가져오기

```
>>> from random import randint
>>> from random import random
>>> randint(1, 100)
82
>>> random()
0.041474812251078186
```

```
>>> from random import randint, random
>>> randint(1, 100)
70
>>> random()
0.004475036264780 63
```

내장라이브러리 사용하기

import

- from 모듈 import *
- 모듈내에 있는 변수와 함수를 전부 불러온다.
- 사용하지 않는 것을 '강력히' 추천한다.
- 많은 모듈을 사용시 어떤 코드가 어떤 모듈에 들어 있는지 파악하기 어렵고, 내부에서 같은 이름을 쓸 경우 오류가 발생할 수 있다.

내장라이브러리 사용하기

import

- import 모듈 as 별칭
- 모듈을 자주 사용하면, 전체 이름을 쓰는 것이 귀찮기 때문에 별칭을 사용하여 import한다.
- 관례적인 별칭이 존재(matplotlib.pylab -> plt, tensorflow -> tf)
- ex) tensorflow를 tf로 import

```
>>> import tensorflow as tf
>>> tf.__version__
'1.8.0'
```

내장라이브러리 사용하기

random모듈

함수	설명
randint(최소(int), 최대(int))	최소이상 최대이하 정수 중 임의의 정수를 반환
random()	0에서 1사이의 부동소수점(float) 숫자를 반환
randrange(시작, 끝[, 간격])	시작숫자이상 끝숫자미만의 (지정된 간격으로 나열된) 숫자 중 임의의 정수를 반환
uniform(최소(float), 최대(float))	최소에서 최대사이에서 임의의 부동소수점을 반환

기본과제-자판기3

vending_machine.py

- 고객이 새로운 기능을 요청하였다. 이전의 프로그램에 추가한다.
- 1. 물품번호를 입력할 때 음수를 입력하면 프로그램이 종료된다. 종료될 때, 돈을 반환한다.
- 2. 1.의 방법이 아니면 프로그램이 계속 동작한다. (무한 루프를 사용)
- 3. 처음 시작시, 물품의 목록을 출력하고 마지막 번호보다 바로 앞 번호는 '돈 넣기'이고 마지막 번호는 거스름돈 반환을 출력하도록 한다.

기본과제-자판기3

- 4. 물품이 나온 이후, 남은 돈이 자판기가 가진 물품의 최소 가격보다 적으면 반환하고 넣은 돈을 0원으로 만든다.
- 5. 물품보다 넣은 돈이 적으면, '돈이 부족합니다. 돈을 더 넣어주세요.'를 출력한다.
- 6. 한 번의 입력에 대한 처리가 끝나면 물품리스트와 돈넣기, 거스름돈 반환을 다시 출력한다.

※ min(반복가능한 자료형), max(반복가능한 자료형)은 각각 자료형 내의 최소값, 최대값을 반환한다.

기본과제-자판기3

※ 어려우면 점프 투 파이썬의 while문 직접만들기를 참고해봅시다!

기본과제-자판기3

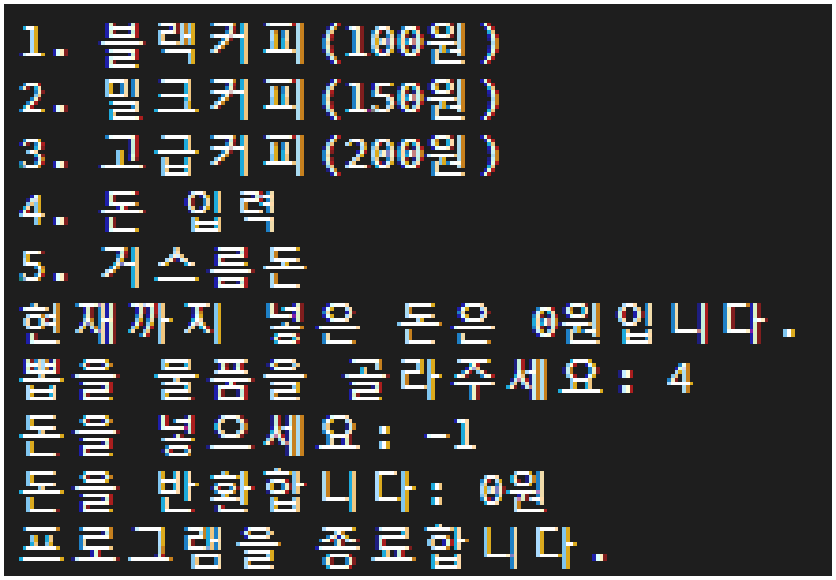
예시

```
1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
현재까지 넣은 돈은 0원입니다.
뽑을 물품을 골라주세요: 3
돈이 부족합니다. 돈을 더 넣어주세요.

1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
현재까지 넣은 돈은 0원입니다.
뽑을 물품을 골라주세요: 4
돈을 넣으세요: 3000
```

```
1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
현재까지 넣은 돈은 2900원입니다.
뽑을 물품을 골라주세요: 3
고급커피이/가 나왔습니다.

1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
현재까지 넣은 돈은 2700원입니다.
뽑을 물품을 골라주세요: 5
돈을 반환합니다: 2700원
```

기본과제-자판기3

예시


1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
현재까지 넣은 돈은 0원입니다.
뽑을 물품을 골라주세요: 4
돈을 넣으세요: -1
돈을 반환합니다: 0원
프로그램을 종료합니다.

기본과제-별트리

startree.py

- 다음과 같은 별 트리를 그리시오.

그리고 싶은 별 트리의 줄 수를 입력하세요(1~79): 4

```
   *(공백 3개)
  ***(공백 2개)
 *****(공백 1개)
*******(공백 0개)
```

기본과제-별트리

startree.py

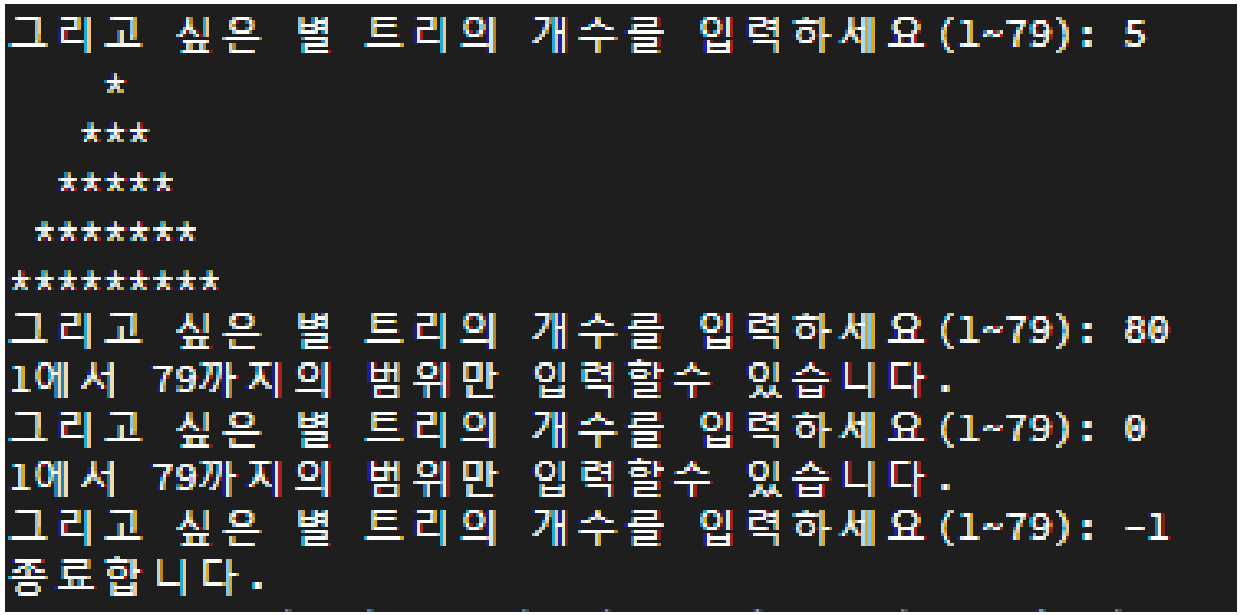
- 조건
  - 입력받는 수는 1-79까지이고, 정수만 입력받는다.
  - 0과 80이상의 정수를 입력받을 경우 "1에서 79까지의 범위만 입력할 수 있습니다."를 출력한 뒤, 다시 맨 처음으로 돌아간다.
  - 음수가 입력될 경우 "종료합니다."를 출력하고 종료한다.

※ print의 개행을 없애기 위해서는 end=""를 사용한다. 기억이 나지 않으면 2차시의 강의자료의 출력을 확인한다.

기본과제-별트리

startree.py

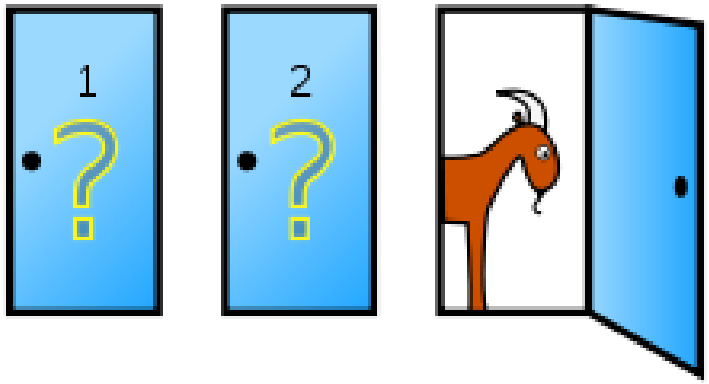
- 예시



심화과제-몬티홀 시뮬레이션

montyhall.py

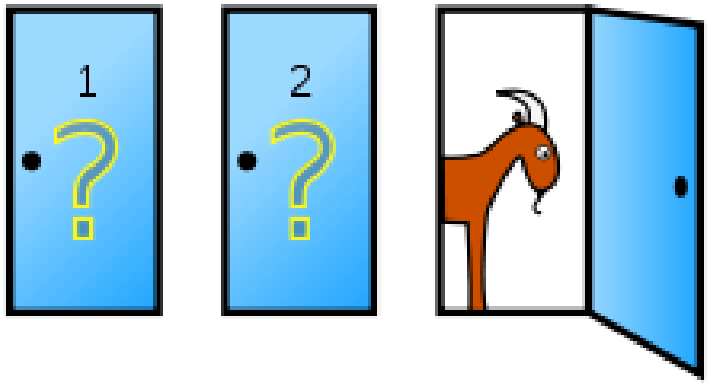
- 세 개의 문 중에 하나를 선택하여 문 뒤에 있는 선물을 가질 수 있는 게임이다.
- 하나의 문에는 자동차가 있고, 나머지 두 문 뒤에는 염소가 있다.



심화과제-몬티홀 시뮬레이션

montyhall.py

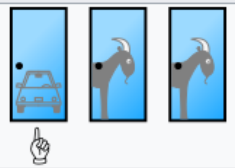
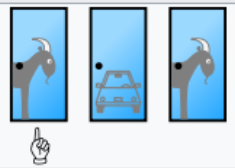
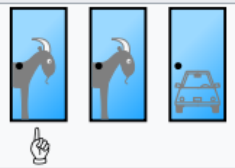
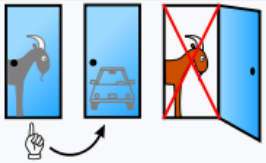
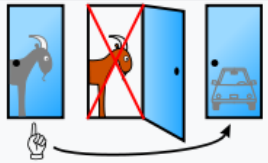
- 참가자가 하나의 문을 선택했을 때, 게임쇼 진행자는 염소가 있는 문을 열어 염소를 보여준다.
- 이때 참가자가 선택했던 번호를 바꾸는 것이 자동차를 얻는데 유리한가?



심화과제-몬티홀 시뮬레이션

montyhall.py

- 바꾸는게 더 유리한데, 항상 열심히 싸우는 문제입니다. 조건부 확률을 사용하면 간단히 나옵니다.

자동차가 1번 뒤에 있을 때		자동차가 2번 뒤에 있을 때	자동차가 3번 뒤에 있을 때
참가자가 1번 문을 선택했을 때			
			
참가자가 1번 문을 선택했고 자동차가 1번 문 뒤에 있을 때		참가자가 1번 문을 선택했고 자동차가 2번 문 뒤에 있을 때	참가자가 1번 문을 선택했고 자동차가 3번 문 뒤에 있을 때
사회자는 두 문 모두 열 수 있다.		사회자는 3번 문을 열 수밖에 없다.	사회자는 2번 문을 열 수밖에 없다.
			
참가자가 1번 문을 선택하고 사회자가 2번 문을 열었을 때	참가자가 1번 문을 선택하고 사회자가 3번 문을 열었을 때	참가자가 1번 문을 선택하였고 자동차가 2번 문 뒤에 있기에 사회자는 3번 문을 열 수밖에 없다	참가자가 1번 문을 선택하였고 자동차가 3번 문 뒤에 있기에 사회자는 2번 문을 열 수밖에 없다
선택을 바꿔서 꽝 1/6	선택을 바꿔서 꽝 1/6	선택을 바꿔서 당첨 1/3	선택을 바꿔서 당첨 1/3
선택을 바꿔서 꽝 1/3		선택을 바꿔서 당첨 2/3	

심화과제-몬티홀 시뮬레이션

montyhall.py

- 몬티홀 문제를 시뮬레이션합니다.
- 조건
- 1. 몬티홀 시뮬레이션을 할 횟수를 입력받는다.
- 2. 바꾸었을 때와 안 바꾸었을 때를 시뮬레이션 하고, 각각 이긴 횟수와 진 횟수를 출력한다.
- 3. 총 횟수에서 이긴 횟수와 진 횟수의 비율이 2/3과 1/3에 가까운지 확인한다.

심화과제-몬티홀 시뮬레이션

montyhall.py

- 확장 문제 조건
  - 문의 총 개수는 n개이고, 시작할 때 문의 개수를 입력받는다.
  - 사회자가 열 수 있는 문의 개수는 k개이고, 시작할 때, 사회자가 여는 문의 개수를 입력받는다.
  - 참여자가 고르는 문의 개수는 1개이다.

  단, 사회자가 열 수 있는 문의 개수는 n-2개까지이다.

- 이론적 답은 (n-1)/n(n-k-1)로 알려져있다.

심화과제-몬티홀 시뮬레이션

montyhall.py

- 예시

```
총 문의 개수를 입력하세요: 3
사회자가 여는 문의 개수를 입력하세요: 1
반복횟수를 입력하세요: 100000
10000번째 입니다.
20000번째 입니다.
30000번째 입니다.
40000번째 입니다.
60000번째 입니다.
70000번째 입니다.
90000번째 입니다.
100000번째 입니다.
총 횟수: 100000
바꾸지 않았을 때 이긴 횟수: 33535
바꿨을 때 이긴 횟수: 66465
바꾸지 않았을 때 이길 이론적 확률: 0.3333
바꾸지 않았을 때 이긴 시뮬레이션 확률: 0.3353
바꿨을 때 이길 이론적 확률: 0.6667
바꿨을 때 이긴 시뮬레이션 확률: 0.6646
```

```
총 문의 개수를 입력하세요: 8
사회자가 여는 문의 개수를 입력하세요: 3
반복횟수를 입력하세요: 100000
10000번째 입니다.
30000번째 입니다.
40000번째 입니다.
60000번째 입니다.
70000번째 입니다.
80000번째 입니다.
90000번째 입니다.
100000번째 입니다.
총 횟수: 100000
바꾸지 않았을 때 이긴 횟수: 12352
바꿨을 때 이긴 횟수: 21987
바꾸지 않았을 때 이길 이론적 확률: 0.1250
바꾸지 않았을 때 이긴 시뮬레이션 확률: 0.1235
바꿨을 때 이길 이론적 확률: 0.2188
바꿨을 때 이긴 시뮬레이션 확률: 0.2199
```