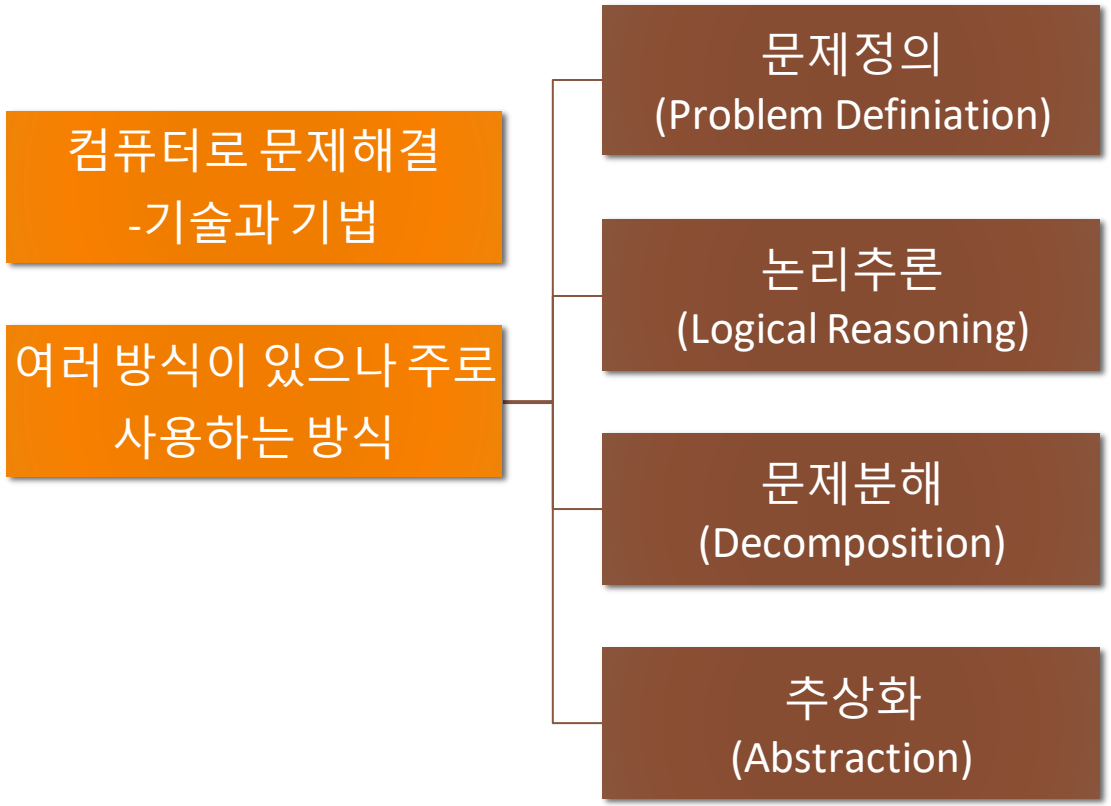
19. 추상화, range₩ 순환문, 함수, ₩ 컴프리핸션

2018.12

일병 김재형

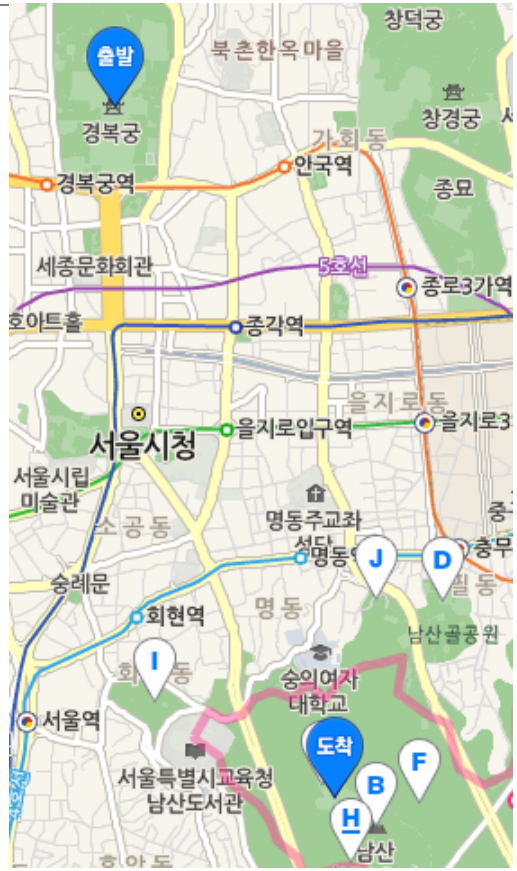
문제해결


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

문제해결4-추상화

추상화(abstraction)

- 문제를 해결할 때 중요한 정보만 선택하여 주어진 상황에만 초점을 맞추는 것이다.
- 예시: 두 지점을 가는 가장 좋은 버스 경로를 찾기

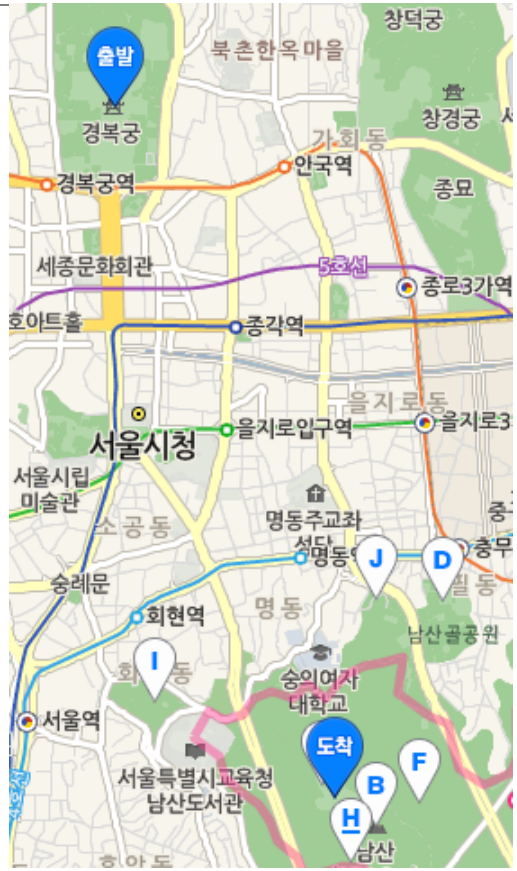


https://www.youtube.com/watch?v=mZkuvP_PsDI

문제해결4-추상화

추상화(abstraction)

- 주변의 관광명소, 지하철역, 택시 승강장등은 중요하지 않다.
- 중요한 것
  버스 번호, 버스 승강장, 버스 도착 시간, 노선의 모양

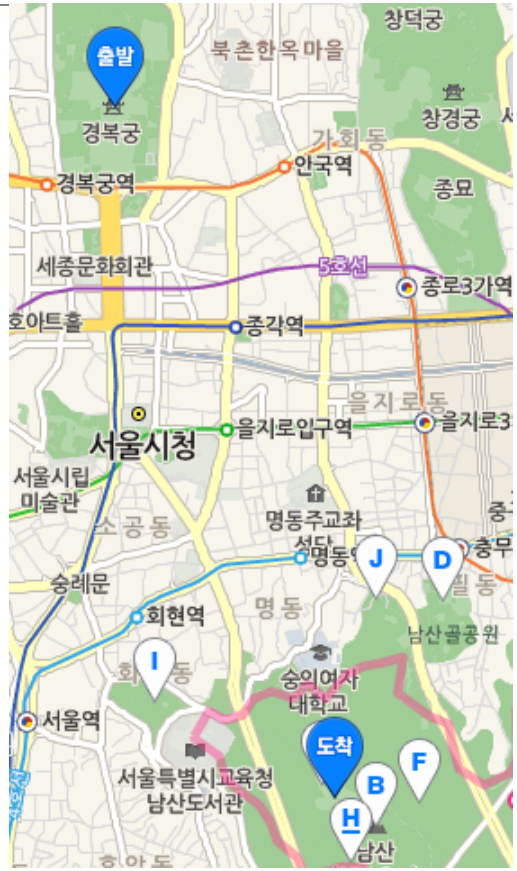


https://www.youtube.com/watch?v=mZkuvP_PsDI

문제해결4-추상화

추상화(abstraction)

- 주변의 관광명소, 지하철역, 택시 승강장등은 중요하지 않다.
- 중요한 것
  버스 번호, 버스 승강장, 버스 도착 시간, 노선의 모양

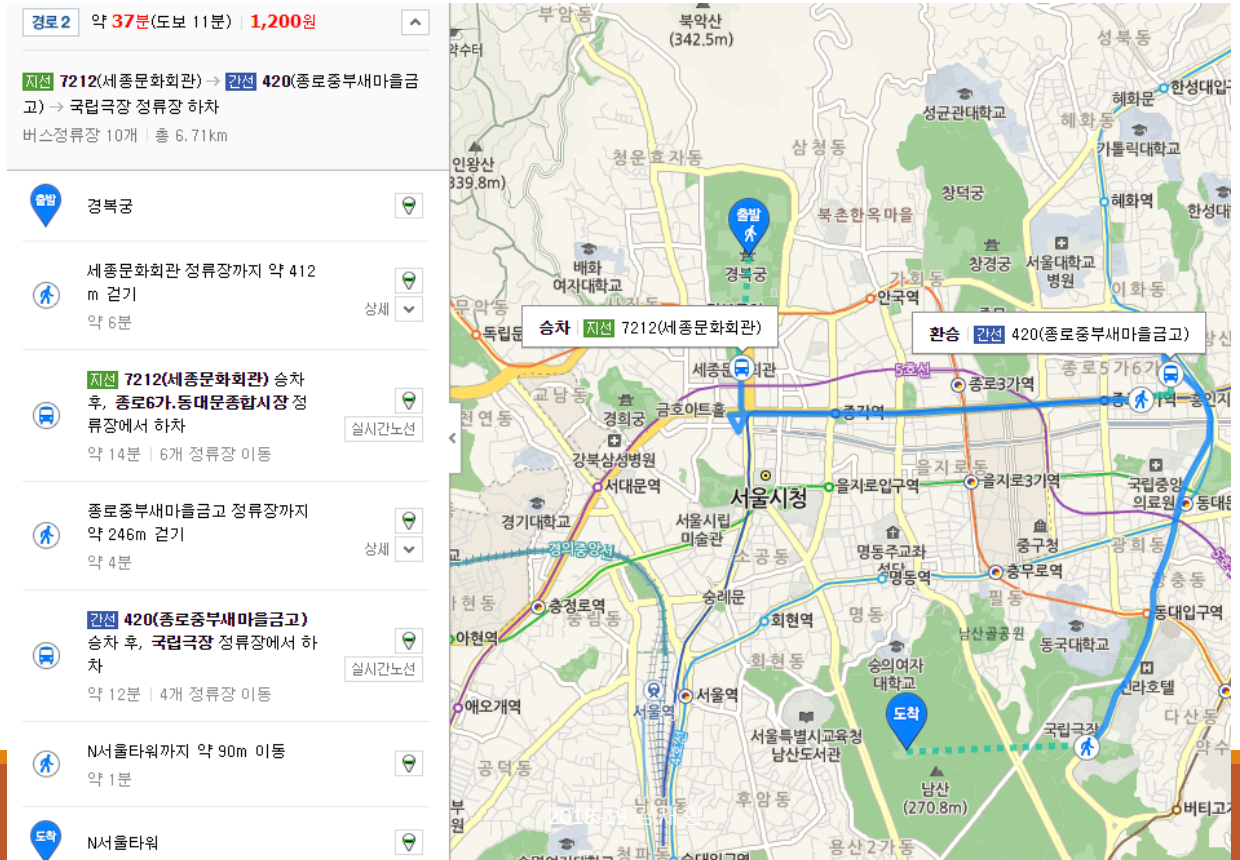


https://www.youtube.com/watch?v=mZkuvP_PsDI

문제해결4-추상화

추상화(abstraction)

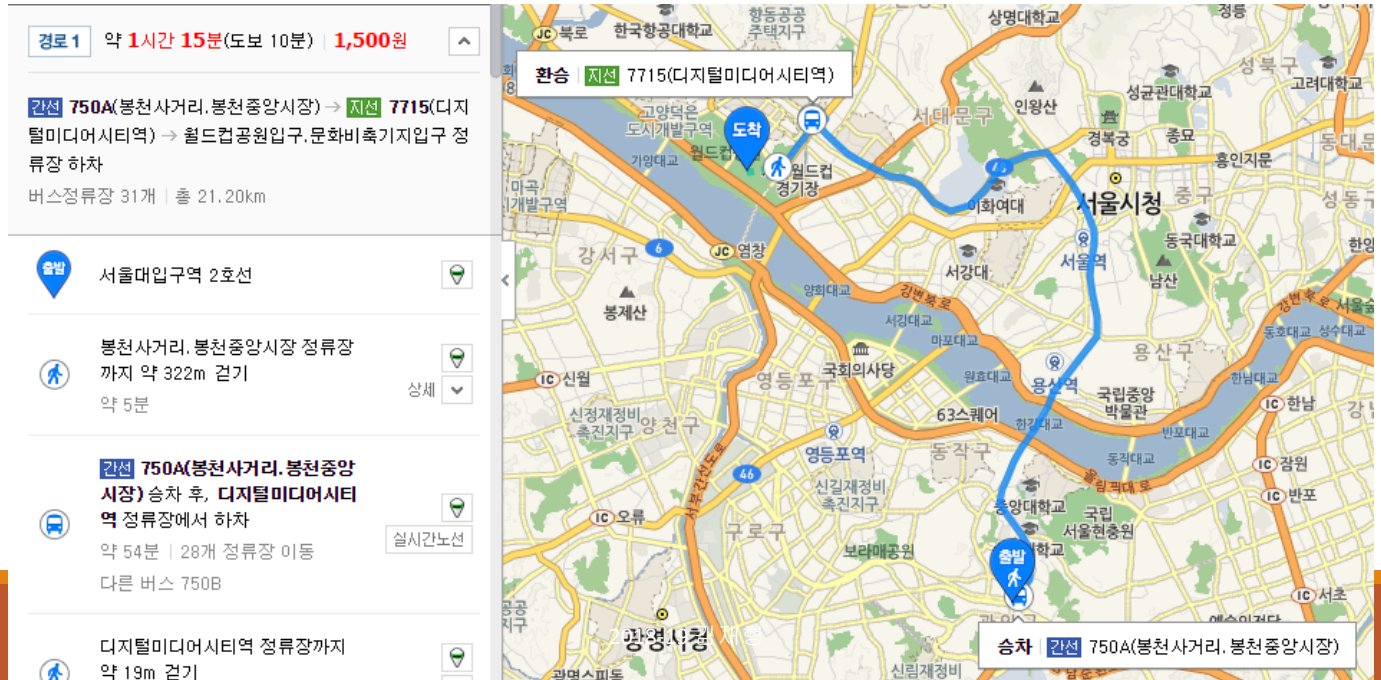
- 버스번호, 버스 승강장, 버스 도착시간, 노선의 모양



문제해결4-추상화

추상화(abstraction)

- 버스번호, 버스 승강장, 버스 도착시간, 노선의 모양
  - 정보들은 예시 작업을 해결 시 필수적이다.
  - 이 네 가지 정보는 다른 경로를 나타낼 때도 사용된다.



문제해결4-추상화

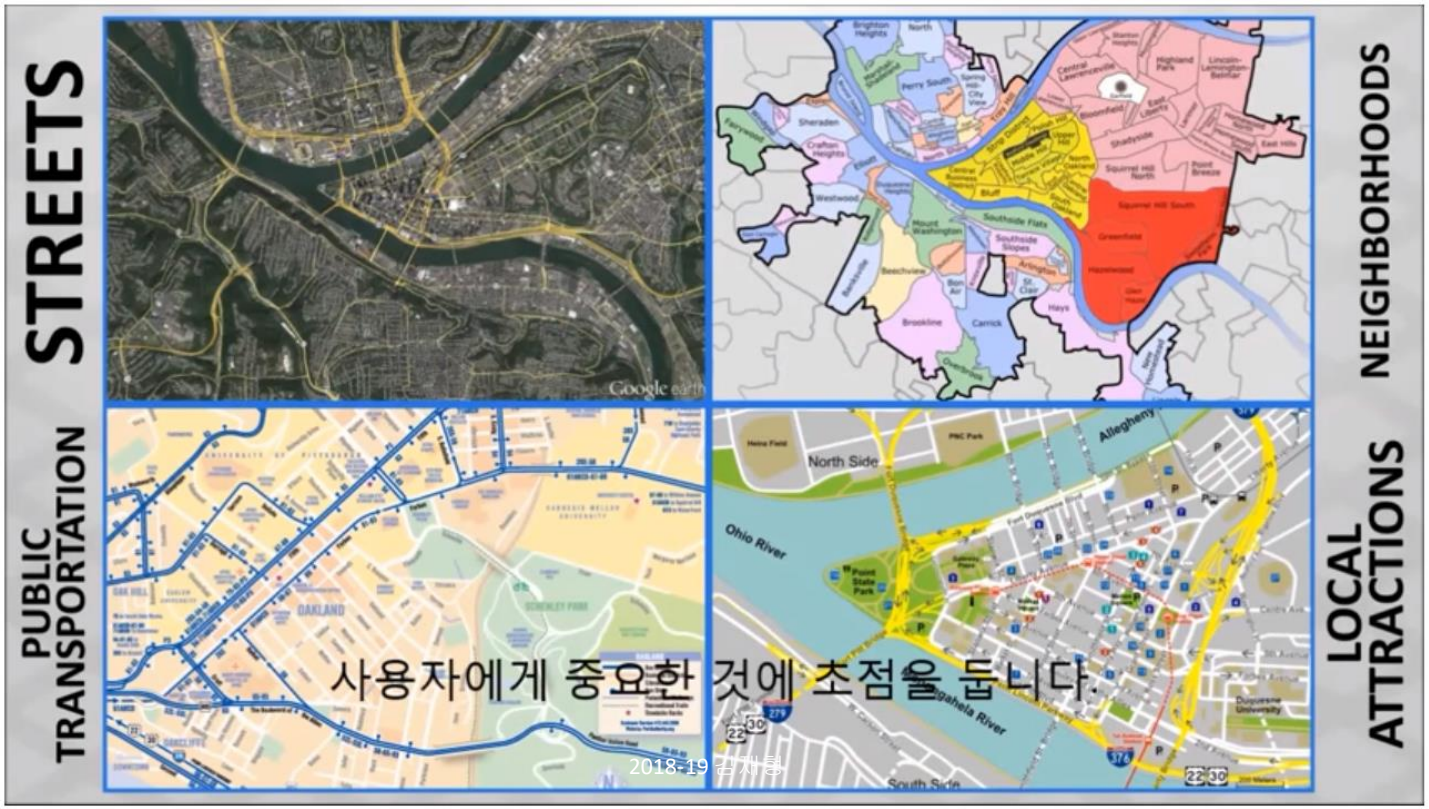
추상화(abstraction)

- 추상화 하나는 비슷한 종류의 전체 클래스를 나타낼 수 있다.
- 여분의 정보가 없어 이해하기 쉽다.

문제해결4-추상화

추상화(abstraction)

- 모든 추상화는 목적을 두고 만들었다.



문제해결4-추상화

추상화(abstraction)

- 제어의 추상화, 자료의 추상화, 행위의 추상화를 통해 문제를 해결한다.

문제해결4-추상화

제어의 추상화(cntrol abstraction)

- 제어구조 (control structure)
  - 명령어가 수행되는 순서를 서술
  - 제어의 추상화는 제어구조의 형태로 표현
- 알고리즘의 5가지 기본 제어구조
  1. 순차제어
  2. 선택
  3. 반복
  4. 제어의 추상화
  5. 병렬처리

문제해결4-추상화

제어구조(Control Structure)

요리법

1. 다음 재료들을 전자레인지에 사용가능한 접시에 넣는다:
   초코렛 칩 3 컵
   압축 우유 14 온스 1컵
   버터 ¼ 컵
2. 원한다면, 1 컵의 호두 조각을 넣어 젓는다.
3. 1분간 전자레인지에서 익힌다.
4. 전자레인지에서 혼합물을 꺼낸 후 다시 젓는다.
5. 초코렛 칩이 완전하게 녹을 때 까지 3단계와 4단계를 반복한다.
6. 바닐라 1 스푼을 혼합물에 넣는다.
7. 혼합물을 8 × 8 크기의 접시에 넣는다.
8. 3 시간동안 냉장시킨다.
9. 퍼지를 1인치 크기의 정사각형으로 자른다.

선택

반복

그림 4.14 퍼지 요리법

문제해결4-추상화

제어의 추상화(control abstraction)

- 알고리즘의 한 명령어가 다른 부분의 알고리즘을 참조할 때 발생
- 추상화를 통해 상세내용을 제거한다.

요리법

1. 퍼지를 만든다(그림 4.14 참조).
2. 초코렛 칩 쿠키를 만든다.
3. 땅콩 버터 바를 만든다.
4. 퍼지, 쿠키, 바 들을 큰 쟁반에 배치한다.

그림 4.15 간식 세트 만들기

문제해결4-추상화

자료(data)의 추상화(data abstraction)

- UML(Unified Modeling Language)
  - 객체지향 소프트웨어의 모델을 만드는 표준 그래픽언어
  - 이 언어는 모델을 나타내는 방법일 뿐이다.
- 클래스 다이어그램
  - 시스템의 정적구조를 나타낸다.
  - 정적구조란, 구성요소와 구성요소사이의 구조적 관계를 나타낸다.

문제해결4-review_클래스

클래스(Class)

- 데이터를 추상화하는 단위
- 실생활의 사물을 소프트웨어로 구현하기 위해서는 추상화(Abstraction, 단순화하는 과정)가 필요.
- 같은 상태와 행위를 가진 객체는 같은 클래스이다.
- 속성(attribute)와 메서드(method)를 가진다.
  - 속성: 객체에 저장된 자료의 특성과 이름을 정의한 코드
  - 메서드: 객체의 행위를 구현한 함수(프로시저)

문제해결4-추상화

자료(data)의 추상화(data abstraction)

- 클래스 다이어그램(Class Diagram)

클래스 이름

속성

동작

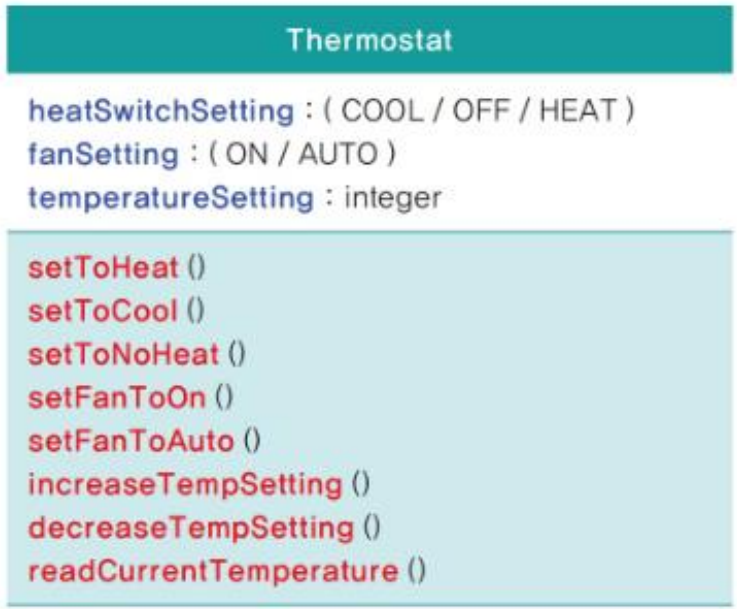


그림 4.16 Thermostat 클래스 다이어그램

문제해결4-추상화

자료(data)의 추상화(data abstraction)

- 클래스 다이어그램(Class Diagram)

클래스 이름

속성

-매개변수로 메서드 줄임

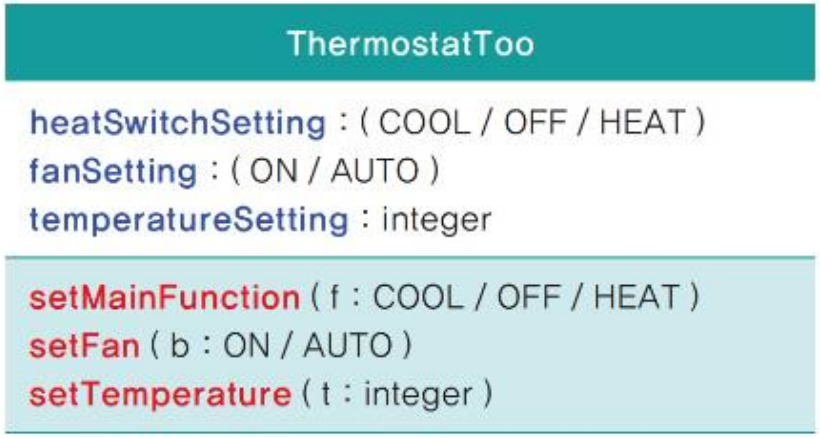


그림 4.17 Thermostat Too 클래스 다이어그램

문제해결4-추상화

행위의 추상화

- 사용 사례 다이어그램(Use Case Diagram)
  - 시스템의 기능을 모델링한다.
  - 행위자(Actor): 같은 종류의 사용자 집단
  - 사용 사례(use case): 행해질 수 있는 행동
  - 행위자와 행위자가 할 수 있는 사용 사례간에는 선으로 연결
  - 행위자는 역할(role)로 분류

문제해결4-추상화

사용 사례 다이어그램
(Use Case Diagram)

- 행위자
  - 고객
  - 점원
- 사용 사례
  - 술구입
  - 물품구입 등

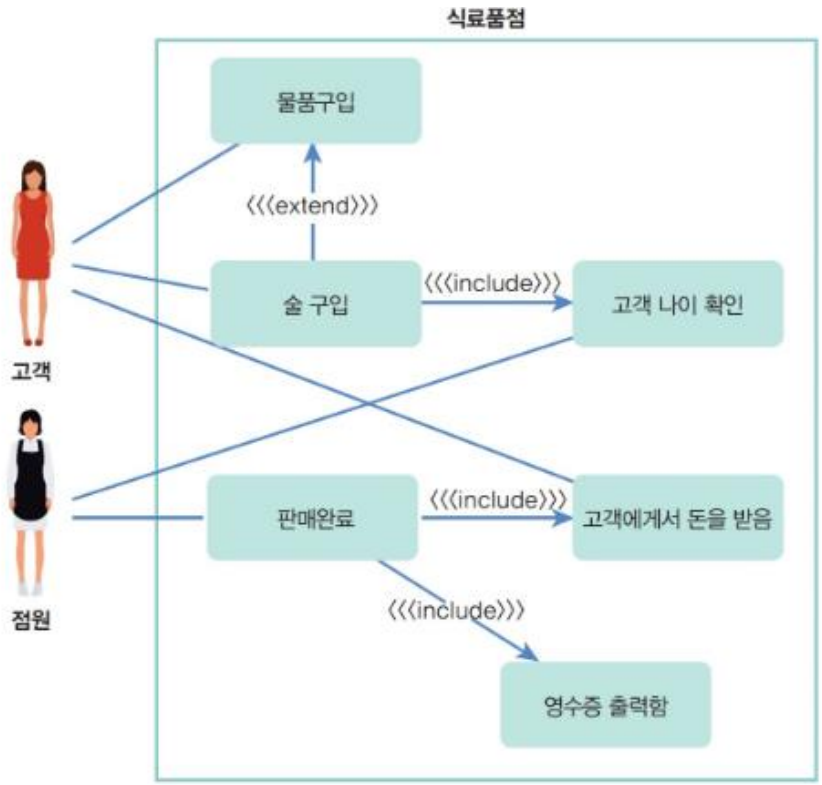


그림 4.21 식료품점 유스 케이스 다이어그램

문제해결4-추상화

사용 사례 다이어그램
(Use Case Diagram)

- <<extended>>
  - 사용 사례가 다른 사용사례의 확장된 사용 사례이거나 특수화된 사용 사례인 경우
- <<included>>
  - 하나의 사용사례가 그 기능의 일부로 다른 행동을 수반하는 경우

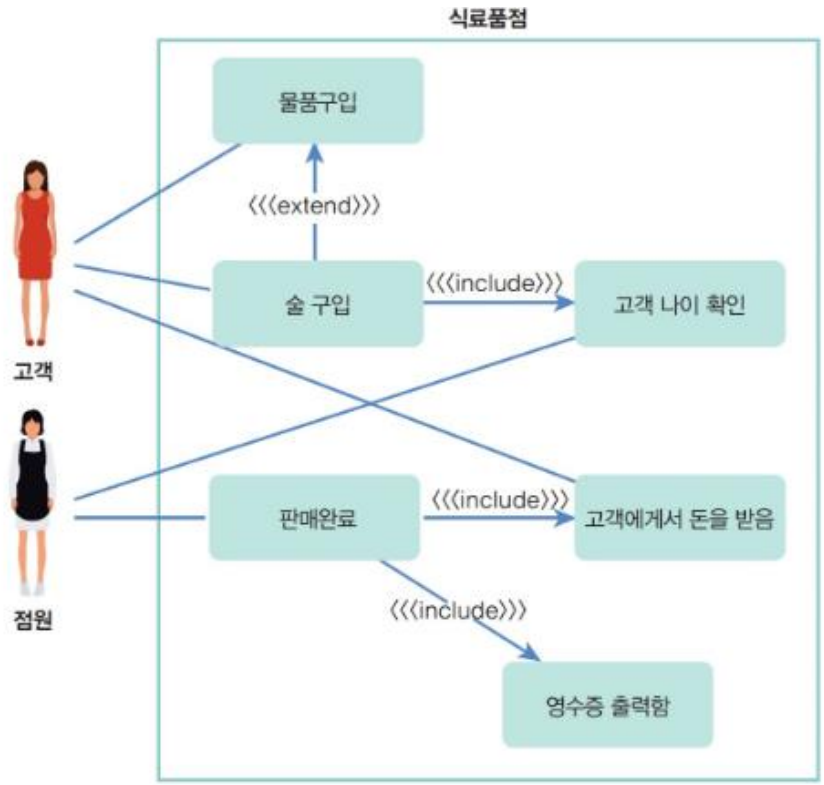


그림 4.21 식료품점 유스 케이스 다이어그램

순환문(for)

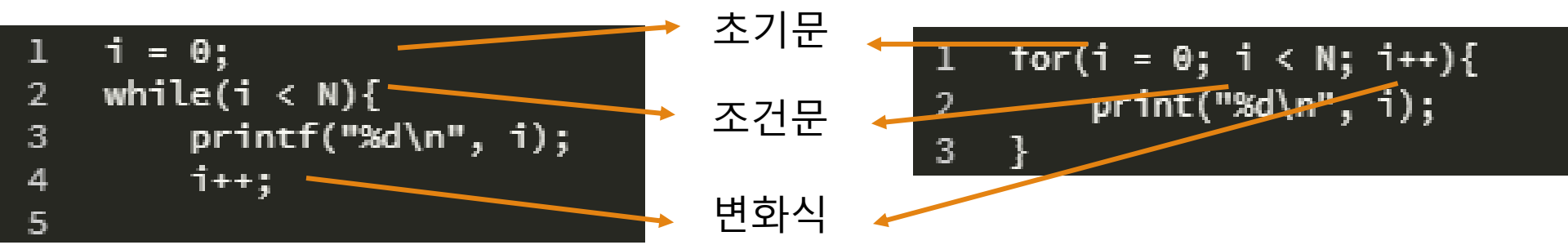
for

- 시작하기 전에: Python에서의 for은 다른 언어의 foreach로 보는게 좋다.

순환문(for)

for(C언어)

- while이 있는데 for은 왜 필요할까?
- '0에서 N 미만의 수에 대해 처리를 한다.'
- while문을 사용시 코드가 분산, 의도이해가 어렵다.
- if문으로 continue를 사용할 때 변화식을 잊기 좋다.



순환문(for)

for(Python)

- Java에서는 확장 for문, Perl, PHP, C#에서는 foreach문이라 불린다.
- Python에서 for문은 '어떤 대상의 요소 전부에 어떤 처리를 한다'로 사용된다.

```
numbers = [1, 2, 3, 4]

i = 0
while i < len(numbers):
    print(numbers[i])
    i += 1
```

```
1  numbers = [1, 2, 3, 4]
2
3  for number in numbers:
4      print(number)
```

순환문(for)

for의 구조

for 변수명 in 순환 가능 객체(리스트, 튜플, 문자열 등)

수행문1

수행문2

순환문(for)

for의 사용예시

```
>>> a = ["Python", "is", "easy"]
>>> for string in a:
...     print(string)
...
Python
is
easy
```

```
>>> a = "abcd"
>>> for charater in a:
...     print(charater)
...
a
b
c
d
```

range()

range([start,] end[, step])

- 해당되는 범위(start-end)의 값을 반복 가능한 객체로 만들어 반환한다.
- range자료형을 반환
- for과 많이 사용된다.

range()

range([start,] end[, step])

- 인수를 한 개 입력(end만 입력)

```
>>> list(range(3))
[0, 1, 2]
```

- 인수를 두 개 입력(start, end만 입력)

```
>>> list(range(1, 5))
[1, 2, 3, 4]
```

- 인수를 세 개 입력(start, end, step 입력)

```
>>> list(range(2, 10, 3))
[2, 5, 8]
```

range()

range()가 숫자가 감소하는 객체를 가지게 하기

- range(4, 0, -1)

※ 4는 포함되고, 0은 포함되지 않는다.

```
>>> list(range(4, 0, -1))
[4, 3, 2, 1]
```

- reversed(range(4))

```
>>> list(reversed(range(4)))
[3, 2, 1, 0]
```

순환문과 range()

배열 순환

```
>>> a = ['a', 'b', 'c', 'd']
>>> for index in range(len(a)):
...     print(a[index])
...
a
b
c
d
```

enumerate()

enumerate(순환 가능 객체)

- '열거하다'라는 뜻이다.
- 시퀀스 자료형을 받아, 각 요소의 숫자를 포함하는 enumberate객체를 반환

```
>>> a = list(['a', 'b', 'c'])
>>> a = ['a', 'b', 'c']
>>> list(enumerate(a))
[(0, 'a'), (1, 'b'), (2, 'c')]
>>> for i, alphabet in enumerate(a):
...     print(i, alphabet)
...
0 a
1 b
2 c
```

순환문(for)

딕셔너리의 순환

- 딕셔너리의 keys 메서드를 통해 순환할 수 있다.
- 딕셔너리의 items 메서드를 통해 순환할 수 있다.

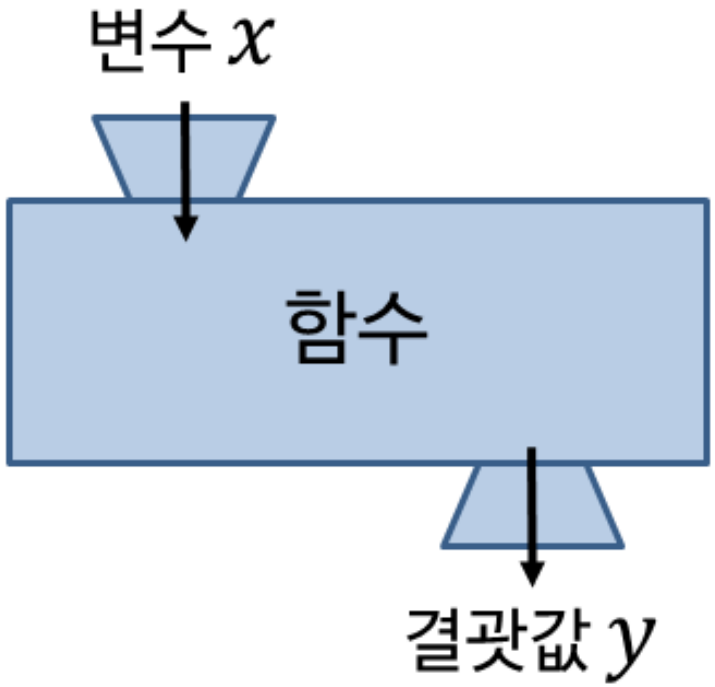
```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> for key in a.keys():
...     print(a[key])
...
1
2
3
```

```
>>> for key, value in a.items():
...     print(key, value)
...
a 1
b 2
c 3
```

함수

함수란?

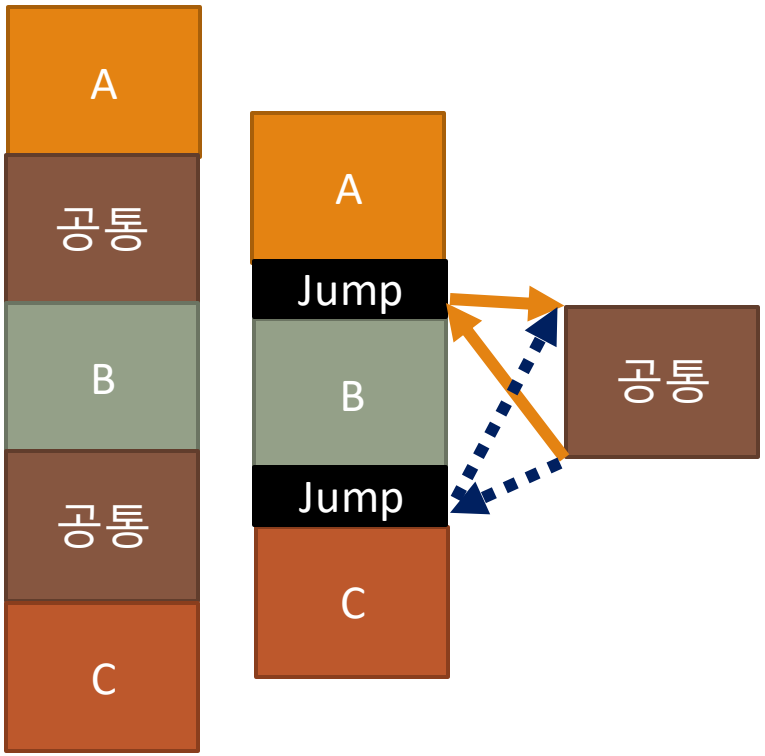
- 어떠한 값(x)를 집어넣었을 때, 결과값(y)가 출력되는 것



함수

함수를 사용하는 이유

- 똑같은 내용(반복되는 내용)이 있다.
- 반복되는 내용을 재사용하기 위해 사용한다.
- 코드 전체가 간결해지고, 이해하기 쉬워진다.



함수

함수의 탄생

- '여러 번 사용하는 명령을 한 곳에 모아 다시 사용한다'는 필요성은 1949년, EDSAC에도 있었다.
- 명령과 데이터가 메모리에 같은 형태로 기록되어 이동이 간단했다.

1: 100번째 명령 목적지변경: 3
2: 함수 호출(jump 80)
3: 다음 명령
……
20: 100번째 명령 목적지변경: 22
21: 함수호출(jump 80)
22: 다음명령
…..
80: 함수처리
……
100: 돌아간다 (0으로 점프)

함수

함수의 탄생

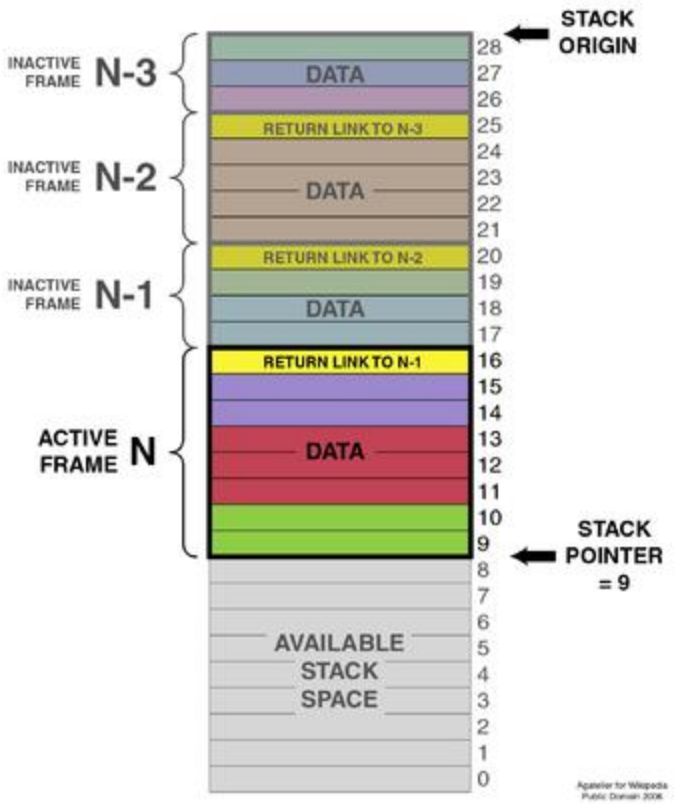
- 이 방식은 사람이 명령 전체를 기억해야 되고, 변경시 전체를 변경해야 한다.
- 돌아갈 목적지를 기억하는 메모리를 만들었다.

1: '돌아갈 목적지 메모리'에 3을 넣는다.
2: 함수 호출(jump 100)
3: 다음 명령
……
80: 함수처리
……
100: 돌아간다 (돌아갈 목적지 메모리를 참조하여 돌아간다.)

함수

함수의 탄생

- 여러 함수를 연속적으로 호출시, 돌아갈 목적지 메모리의 내용이 사라진다.
- 스택(Stack)이 등장



함수

함수의 구조

함수

함수의 결과값은 하나

함수

매개변수(입력인수)

- 위치인자
- 키워드 인자

함수

매개변수(입력인수)

- 위치인자모으기
- 키워드 인자모으기

함수-네임스페이스와 스코프

함수-docstring

컴프리핸션(comprehension)

리스트 컴프리핸션(지능형 리스트, 리스트 내포)

- 새로운 리스트를 만들 때, 보기 좋게 만드는 방법
- [표현식 for 표현식변수 in 순환 가능 객체]
- 예시

```
>>> # 10개의 3의 배수 요소를 가진 리스트 만들기
...
>>> result = [num * 3 for num in range(10)]
>>> result
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

컴프리핸션(comprehension)

리스트 컴프리핸션(지능형 리스트, 리스트 내포)

- 조건을 추가하기
- [표현식 for 표현식변수 in 순환 가능 객체 if 조건]
- 조건에 맞는 시퀀스 자료형 내의 요소가 표현식 변수로 넘어간다.

```
>>> # 짝수에만 3을 곱해 리스트 만들기
...
>>> result = [num * 3 for num in range(10) if num % 2 == 0]
>>> result
[0, 6, 12, 18, 24]
```

컴프리핸션(comprehension)

리스트 컴프리핸션(지능형 리스트, 리스트 내포)

- 표현식변수를 추가하기
- [표현식 for 표현식변수1 in 순환 가능 객체
  for 표현식변수2 in 순환 가능 객체]

```
>>> colors = ['black', 'blue']
>>> sizes = [80, 82, 84]
>>> pants = [(color, size) for color in colors for size in sizes]
>>> pants
[('black', 80), ('black', 82), ('black', 84), ('blue', 80), ('blue', 82), ('blue', 84)]
```

- for color in colors:
  for size in sizes:

컴프리핸션(comprehension)

리스트 컴프리핸션(지능형 리스트, 리스트 내포)

- [표현식 for 표현식변수1 in 순환 가능 객체
  for 표현식변수2 in 순환 가능 객체]
- 데카르트 곱

		S		
		80	82	84
R	'black'	('black', 80)	('black', 82)	('black', 84)
	'blue	('blue', 80)	('blue', 82)	('blue', 84)

R X S

컴프리핸션(comprehension)

리스트 컴프리핸션(지능형 리스트, 리스트 내포)

- [표현식 for 표현식변수1 in 순환 가능 객체
  for 표현식변수2 in 순환 가능 객체]
- 다음도 가능

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> flat = [x for row in matrix for x in row]
>>> flat
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> flat = [x for x in row for row in matrix]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'row' is not defined
```

컴프리핸션(comprehension)

딕셔너리 컴프리핸션(지능형 딕셔너리)

- {키_표현식: 값_표현식 for 표현식 변수 in 순환 가능 객체}
- 예시

```
>>> [("key1", "value1"), ("key2", "value2"), ("key3", "value3")]
[('key1', 'value1'), ('key2', 'value2'), ('key3', 'value3')]
>>> dict_list = [("key1", "value1"), ("key2", "value2"), ("key3", "value3")]
>>> test_dict = {key: value for key, value in dict_list}
>>> test_dict
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

- 리스트와 같게 for의 중첩과 조건도 넣을 수 있다.

컴프리핸션(comprehension)

집합 컴프리핸션(지능형 집합)

- {표현식 for 표현식변수 in 순환 가능 객체}

컴프리핸션(comprehension)

컴프리핸션

- 다중 if와 다중 for도 가능하다.
- 단, 매우 복잡해지기 때문에 일반적으로
  - 조건 두 개
  - 루프 두 개
  - 조건 한 개와 루프 한 개 정도만 쓰는 것을 추천한다.

```
>>> # 행렬의 행의 합이 10이상이고, 3으로 나눠지는 셀을 구한다.
...
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> filter = [x for row in matrix if sum(row) >= 10
...                 for x in row if x % 3 == 0]
>>> filter
[6, 9]
```

컴프리핸션(comprehension)

튜플의 컴프리핸션?

- 제너레이터가 생성된다.
- 심화과정에서 설명한다.

기본과제-구구단

간단하게 2단부터 9단까지 출력해보자.

- times_table.py
- 파일을 실행하면 2단부터 9단씩 출력한다.
- 예시

```
root@goorm:/workspace/PythonSeminar18/Exercise/times_table(master)# python3 times_table.py
2 * 1 =  2  3 * 1 =  3  4 * 1 =  4  5 * 1 =  5  6 * 1 =  6  7 * 1 =  7  8 * 1 =  8  9 * 1 =  9
2 * 2 =  4  3 * 2 =  6  4 * 2 =  8  5 * 2 = 10  6 * 2 = 12  7 * 2 = 14  8 * 2 = 16  9 * 2 = 18
2 * 3 =  6  3 * 3 =  9  4 * 3 = 12  5 * 3 = 15  6 * 3 = 18  7 * 3 = 21  8 * 3 = 24  9 * 3 = 27
2 * 4 =  8  3 * 4 = 12  4 * 4 = 16  5 * 4 = 20  6 * 4 = 24  7 * 4 = 28  8 * 4 = 32  9 * 4 = 36
2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25  6 * 5 = 30  7 * 5 = 35  8 * 5 = 40  9 * 5 = 45
2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36  7 * 6 = 42  8 * 6 = 48  9 * 6 = 54
2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49  8 * 7 = 56  9 * 7 = 63
2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64  9 * 8 = 72
2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

기본과제-369

three_six_nine.py

- 고객이 369를 할 때, 짝을 몇 번하는지 알고 싶다고 프로그램을 주문하였다.
- 1. 입력되는 값은 정수이고, 값의 제한은 없다.
- 2. 1부터 입력받은 수까지 한 칸씩 띄우며 출력하며, 20번 째 수마다 줄바꿈을 한다.
- 3. 숫자에 3, 6, 9가 들어가면 그 횟수만큼 짝을 출력한다.

기본과제-369

three_six_nine.py

- 예시

```
마지막 숫자를 입력하세요: 80
1 2 짝 4 5 짝 7 8 짝 10 11 12 짝 14 15 짝 17 18 짝 20
21 22 짝 24 25 짝 27 28 짝 짝 짝 짝 짝짝 짝 짝 짝짝 짝 짝 짝짝 40
41 42 짝 44 45 짝 47 48 짝 50 51 52 짝 54 55 짝 57 58 짝 짝
짝 짝 짝짝 짝 짝 짝짝 짝 짝 짝짝 70 71 72 짝 74 75 짝 77 78 짝 80
```

기본과제-369

three_six_nine.py

- 확장 문제
- 1. 2의 배수는 '뽁'을 출력한다.
  단, 출력 순서가 맞아야 되며, 공배수일 경우 3을 우선적으로 출력한다.
  예시)
  346 '짝''뽁''짝뽁'

```
마지막 숫자를 입력하세요: 40
1 뽁 짝 뽁 5 뽁짝 7 뽁 짝 10 11 뽁 짝 뽁 15 뽁짝 17 뽁 짝 뽁
뽁 뽁뽁 뽁짝 뽁뽁 뽁 뽁뽁짝 뽁 뽁뽁 뽁짝 짝 짝 짝뽁 짝짝 짝뽁 짝 짝뽁짝 짝 짝뽁 짝짝 뽁
```

기본과제-자판기4

심화과제-달팽이

심화과제-LCD display