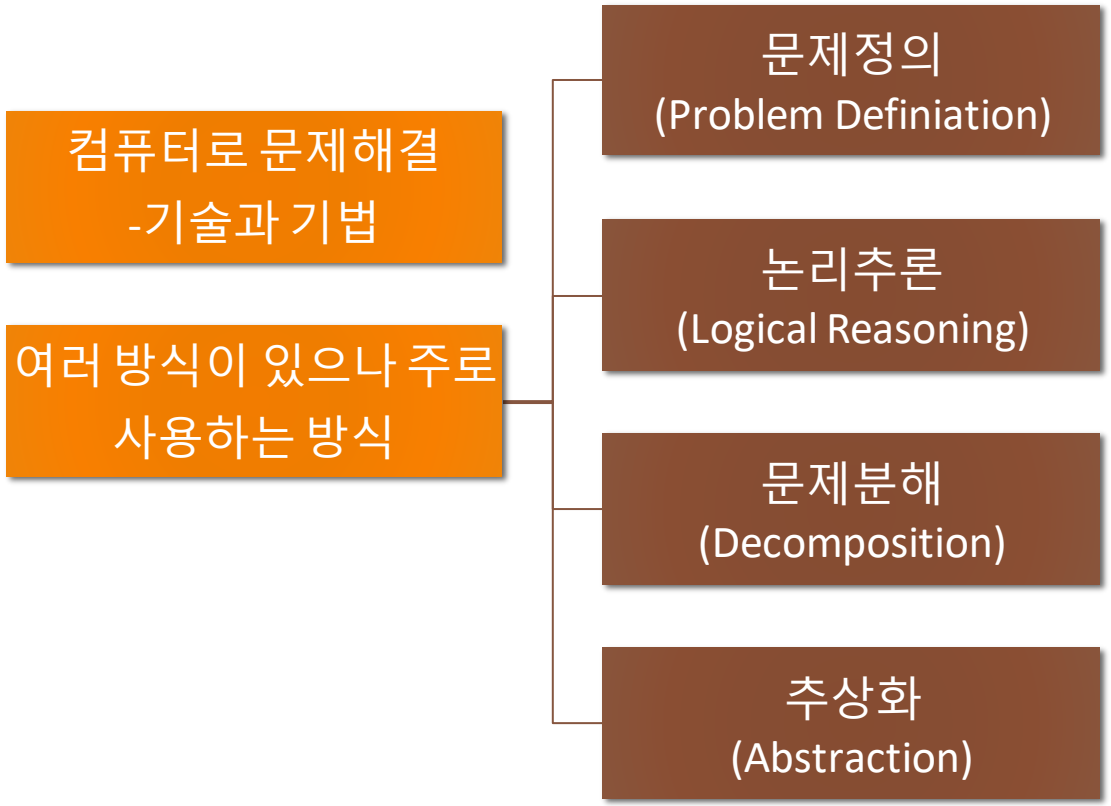
19. 추상화, range₩ 순환문, 함수

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

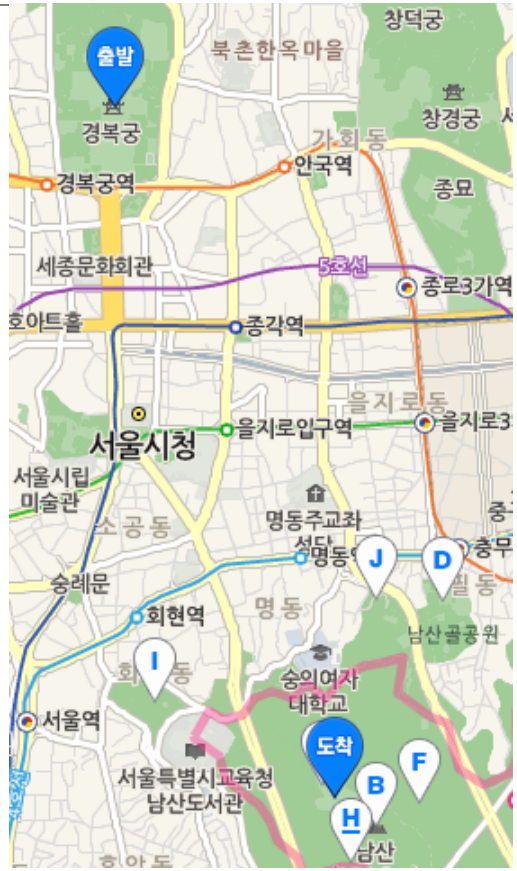
추상화(abstraction)

- 어떤 사물에 대해 중요하지 않거나 세부적인 부분들을 생략하고 중요한 속성에만 집중해서 사물을 묘사하는 방법

문제해결4-추상화

추상화(abstraction)

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

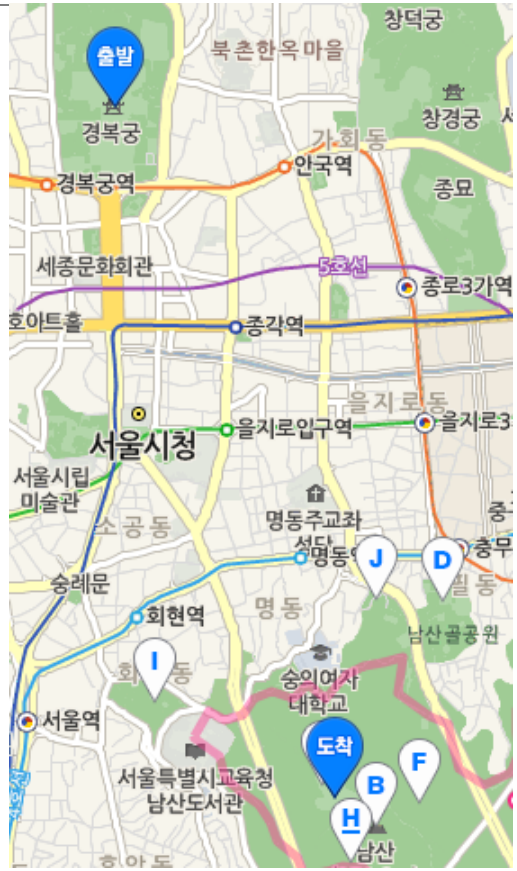


https://www.youtube.com/watch?v=mZkuvP_PsDI

문제해결4-추상화

추상화(abstraction)

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

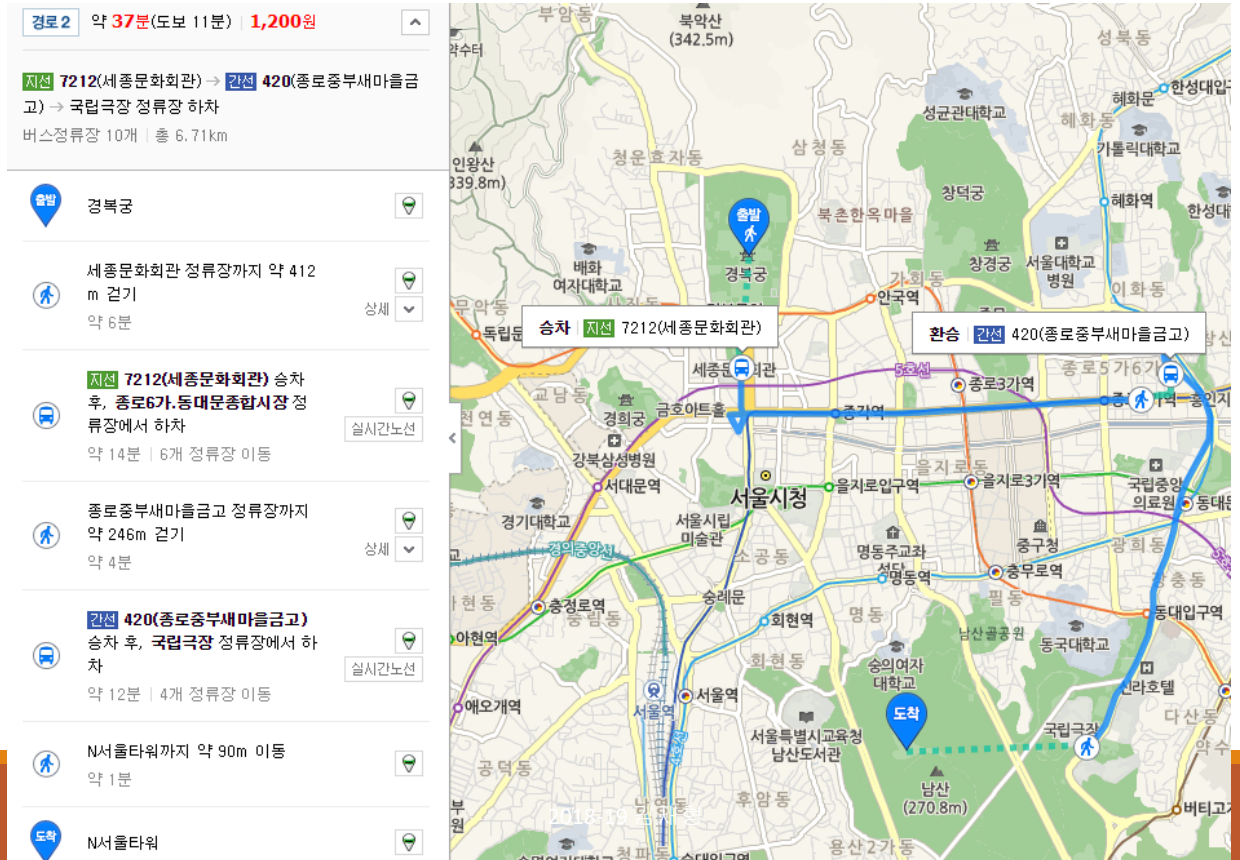


https://www.youtube.com/watch?v=mZkuvP_PsDI

문제해결4-추상화

추상화(abstraction)

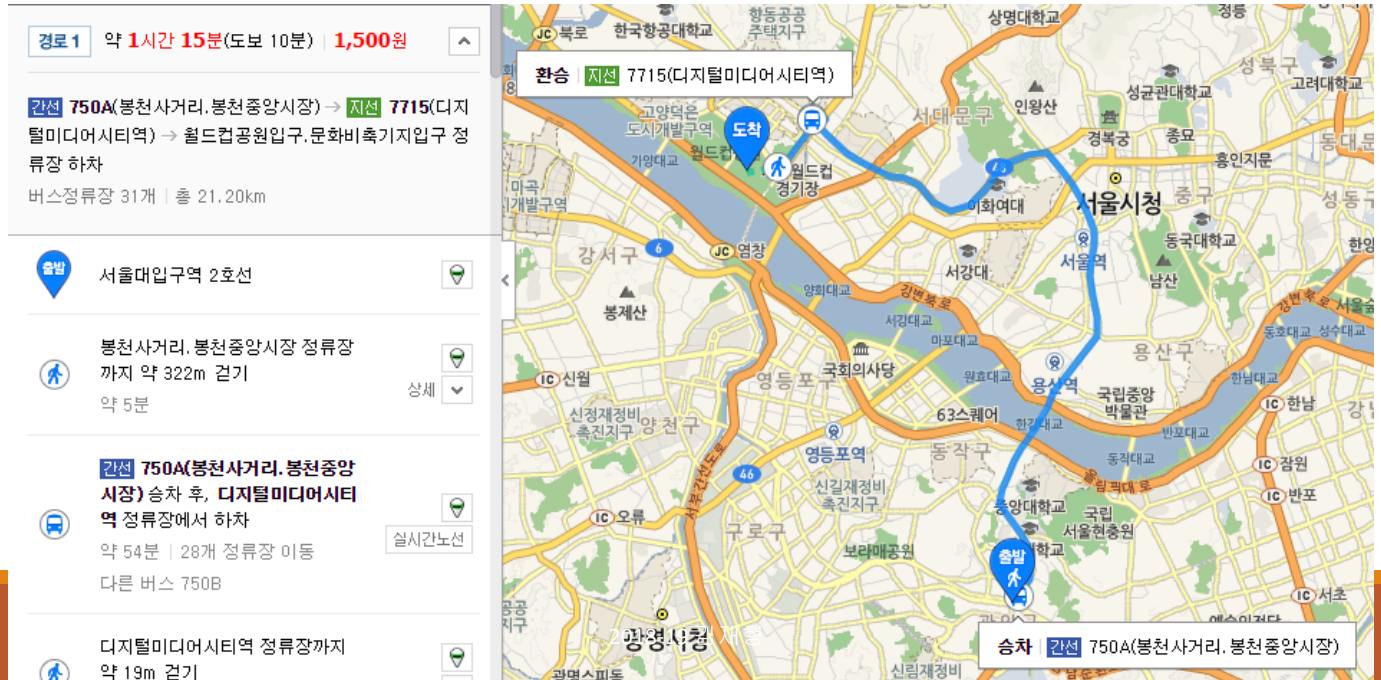
- 버스번호, 버스 승강장, 버스 도착시간, 노선의 모양



문제해결4-추상화

추상화(abstraction)

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



문제해결4-추상화

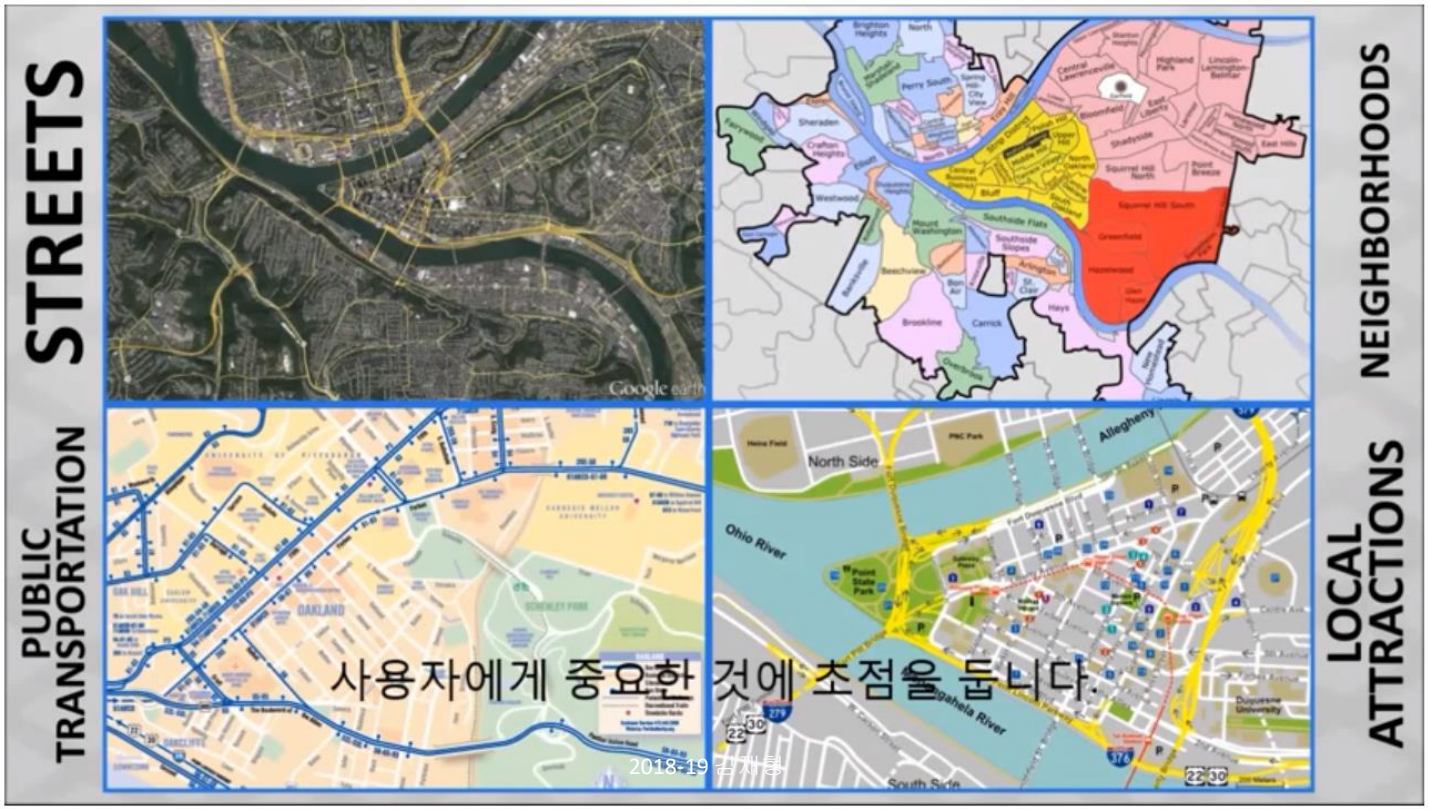
추상화(abstraction)

- 추상화 하나는 비슷한 종류의 전체 집합을 나타낼 수 있다.
- 추가적인 정보가 없어 이해하기 쉽다.

문제해결4-추상화

추상화(abstraction)

- 모든 추상화는 목적을 두고 만들었다.



문제해결4-추상화

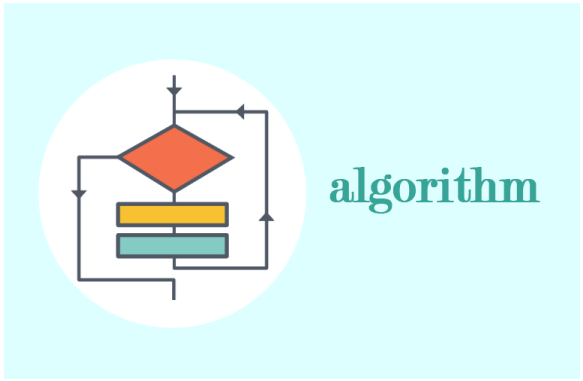
추상화(abstraction)

- 컴퓨터 과학에서 사용되는 추상화
  - 제어의 추상화
  - 자료의 추상화
  - 행위의 추상화

문제해결4-추상화

제어의 추상화(control abstraction)

- 제어구조 (control structure)
  - 명령어가 수행되는 순서를 서술하는 체계
- 알고리즘
  - 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화



문제해결4-추상화

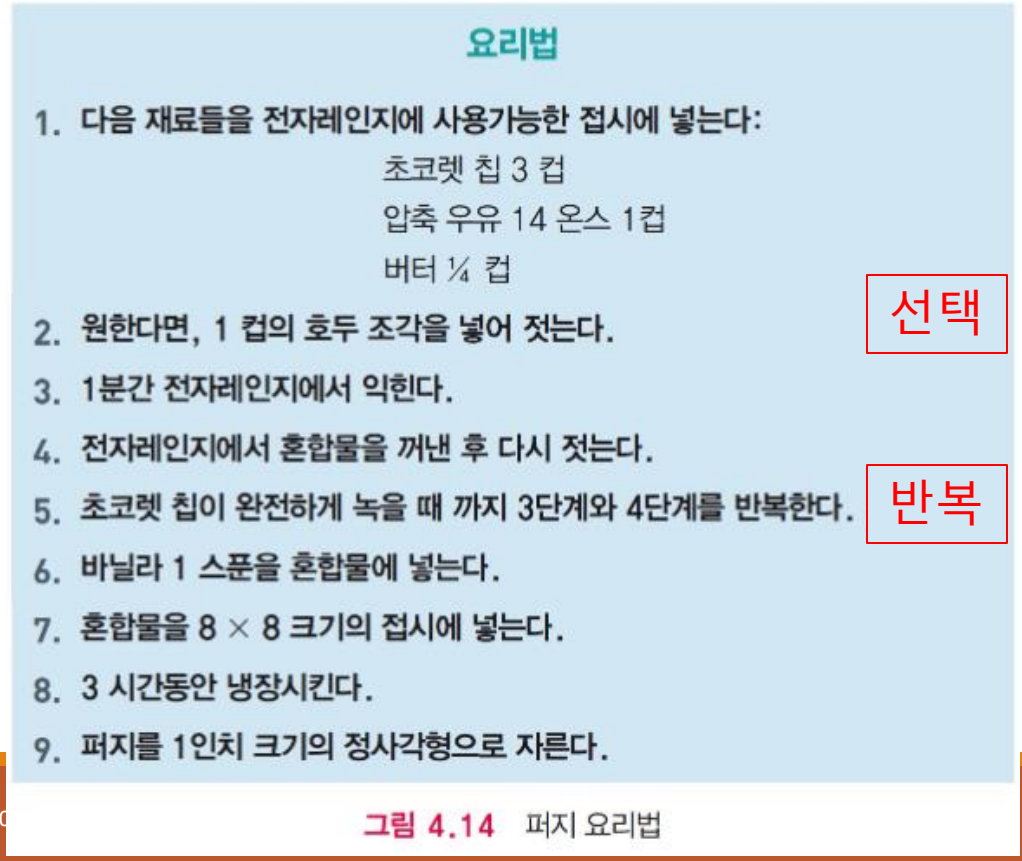
제어의 추상화(control abstraction)

- 알고리즘의 제어구조(아래의 구조를 혼용)
  1. 순차제어: 명령어를 순서대로 수행
  2. 선택: 선택사항 중 하나를 선택
  3. 반복: 명령어를 반복하도록 함
  4. 제어의 추상화: 알고리즘의 명령어가 다른 부분의 알고리즘 참조
  5. 병렬처리

문제해결4-추상화

제어흐름(Control flow)

- 명령어가 수행되는 순서
- 제어 구조를 혼용



문제해결4-추상화

제어의 추상화(control abstraction)

- 알고리즘의 한 명령어가 다른 부분의 알고리즘을 참조
- 이를 통해 산만할 수 있는 상세내용을 제거

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

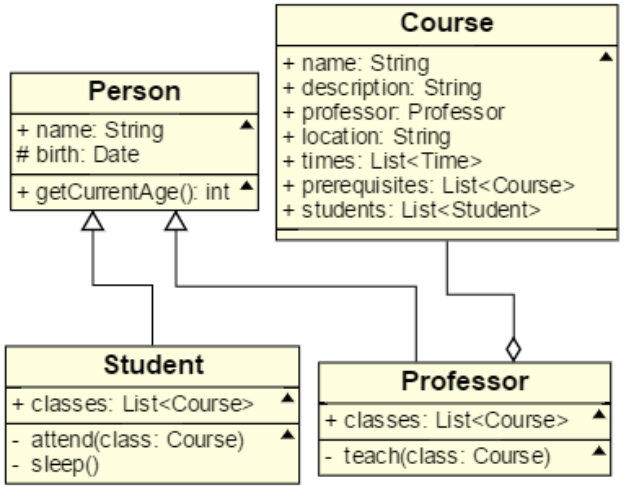
자료의 추상화(data abstraction)

- 자료의 내용 중 필요한 부분만 남겨 한 눈에 알아볼 수 있게 만드는 것

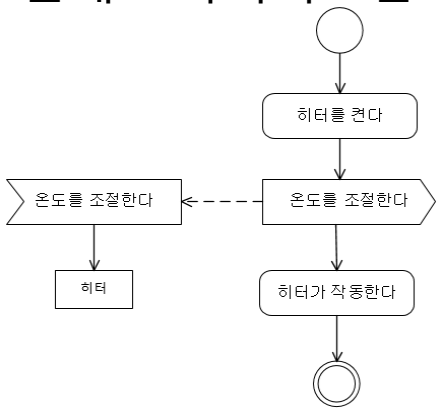
문제해결4-추상화

자료의 추상화
(data abstraction)

- UML(Unified Modeling Language)
  - 객체지향 소프트웨어의 모델을 만드는 표준 그래픽언어
  - 이 언어를 통해 여러 모델을 표현할 수 있다.
  - 이 내부에 클래스 다이어그램이 존재



클래스 다이어그램

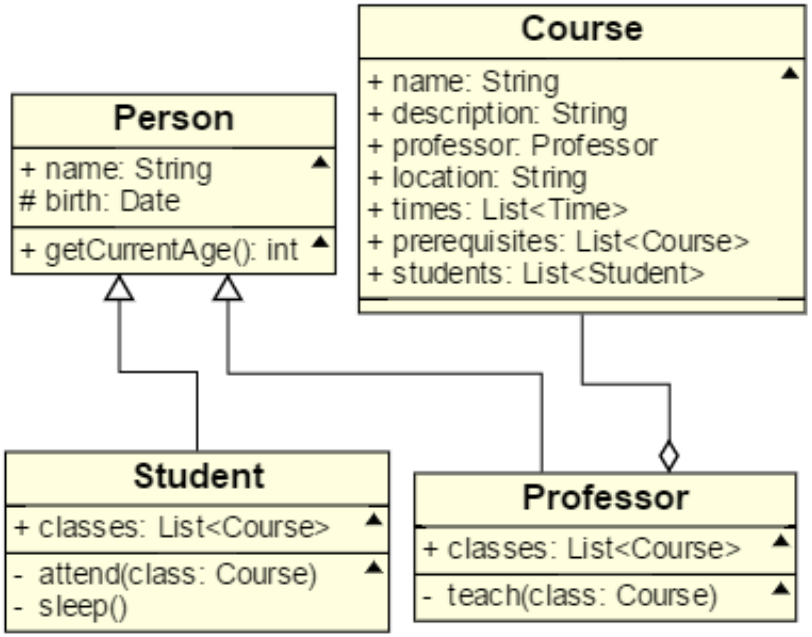


활동 다이어그램

문제해결4-추상화

자료의 추상화
(data abstraction)

- 클래스 다이어그램
  - 시스템의 정적구조를 나타냄
  - 정적구조란, 구성요소와 구성요소사이의 구조적 관계를 나타내는 것
  - 속성과 메서드를 추상화



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

자료의 추상화(data abstraction)

- 클래스 다이어그램(Class Diagram)-자동조절기
  - 속성: 세 가지로 추상화
  - 메서드(동작): 행동을 추상화. 속성을 변경

클래스 이름

속성

동작

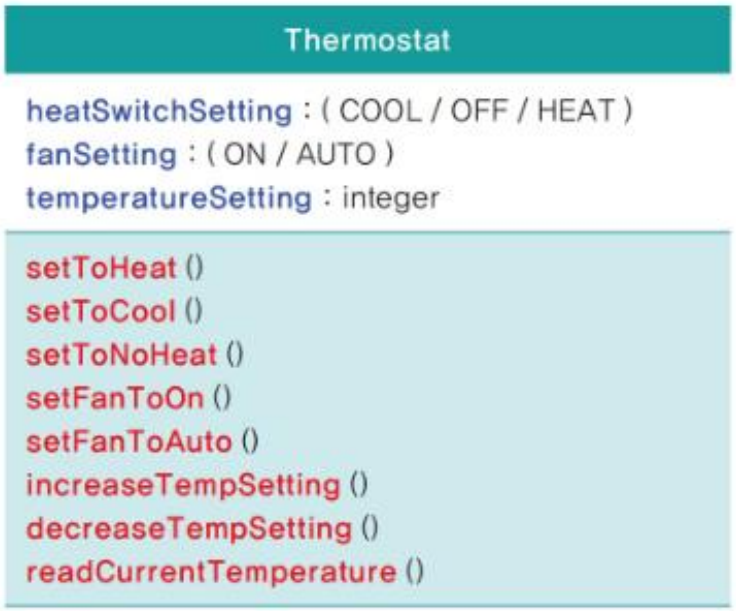


그림 4.16 Thermostat 클래스 다이어그램

문제해결4-추상화

자료의 추상화(data abstraction)

- 클래스 다이어그램(Class Diagram)-자동조절기
  - 메서드에 매개변수를 넣어 같은 행동(메서드)로 속성을 조절할 수 있게 함
  - 유연성을 통해 메서드 감소

클래스 이름

속성

-매개변수로 메서드 줄임

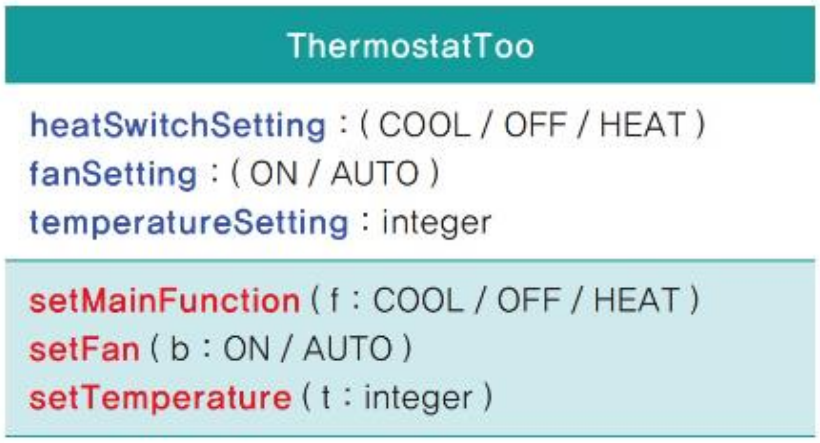
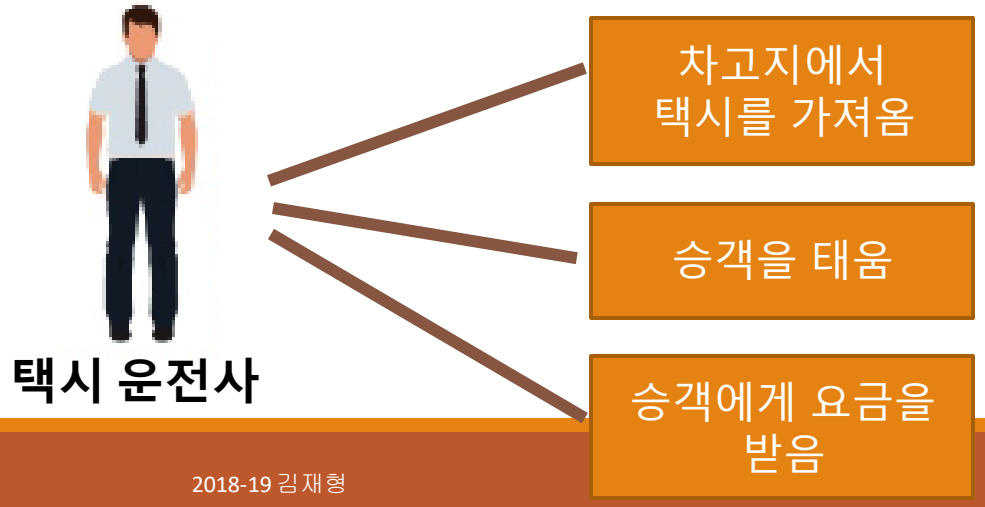


그림 4.17 Thermostat Too 클래스 다이어그램

문제해결4-추상화

행위의 추상화

- 사용 사례 다이어그램(Use Case Diagram)
  - 시스템을 사용자와 시스템 간의 관계로 표현
  - 행위자(Actor): 같은 종류의 사용자 집단
  - 사용 사례(use case): 행해질 수 있는 행동
  - 행위자와 행위자가 할 수 있는 사용사례 간에는 선으로 연결



문제해결4-추상화

사용 사례 다이어그램

- 행위자는 역할(role)로 분류
  - 공통된 행동으로 사용자를 분류하기 위함
- 행위자
  - 학생, 담당직원, 교수 등
  - 행동에 따라 역할이 정해짐
- 사용 사례
  - 교과목 등록, 로그인 등

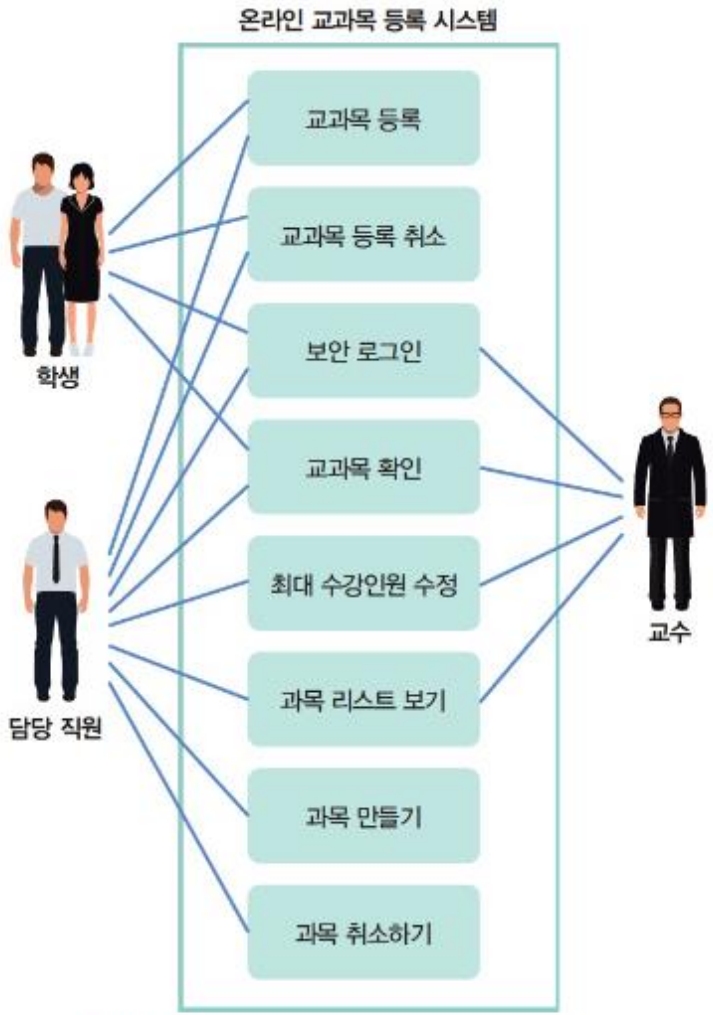


그림 4.20 온라인 교과목 등록 시스템 유스 케이스

문제해결4-추상화

사용 사례 다이어그램

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

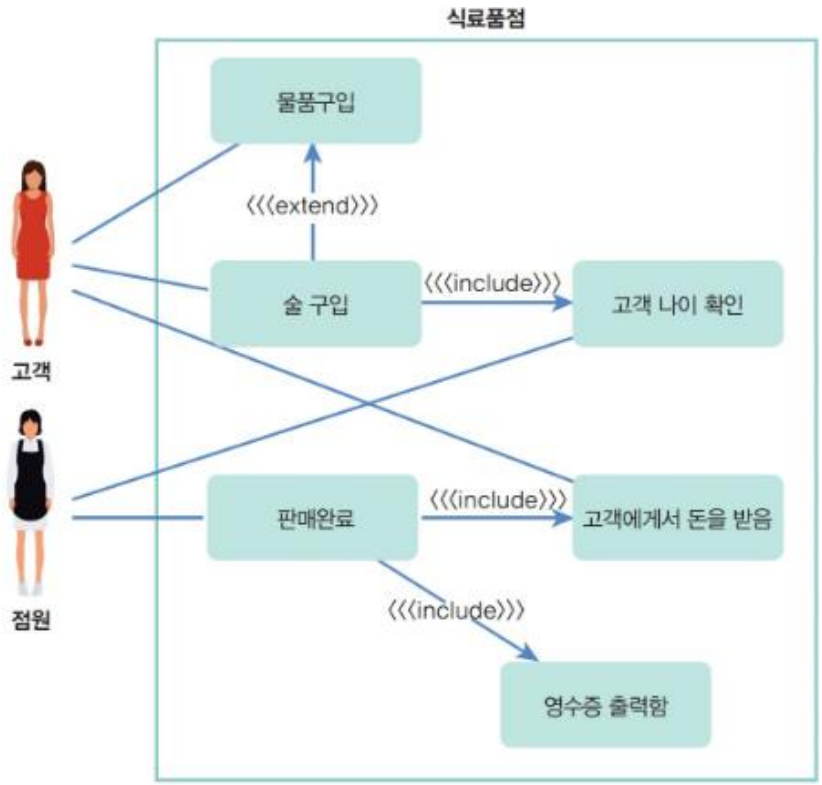


그림 4.21 식료품점 유스 케이스 다이어그램

순환문(for)

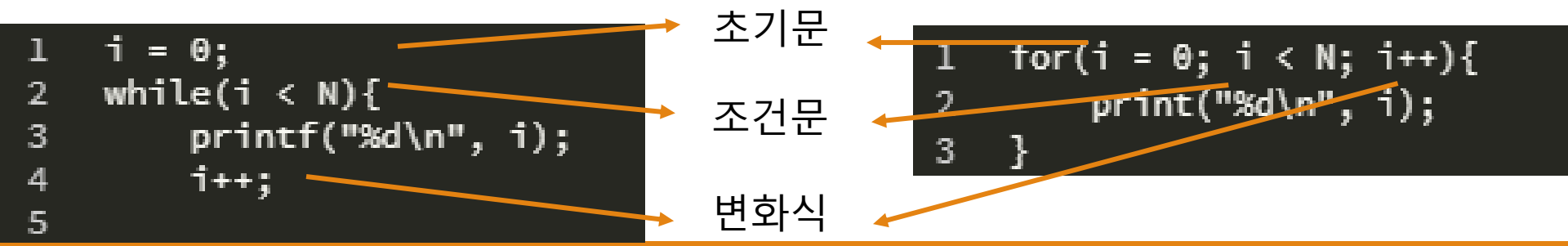
for

- 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 및 의도를 이하하기 어렵다.
  - if문으로 continue를 사용할 때 변화식을 넣는 것을 잊기 쉽다.



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.
- 순환 가능 객체를 받아, 각 요소의 숫자를 포함하는 enumerate객체를 반환

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

enumerate()

enumerate(순환 가능 객체, start=0)

- (순환 가능 객체, 시작번호)를 입력하면 시작번호부터 시작한다.

```
>>> a = ['a', 'b', 'c']
>>> for i, alphabet in enumerate(a, 3):
...     print(i, alphabet)
...
3 a
4 b
5 c
```

순환문(for)

딕셔너리의 순환

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

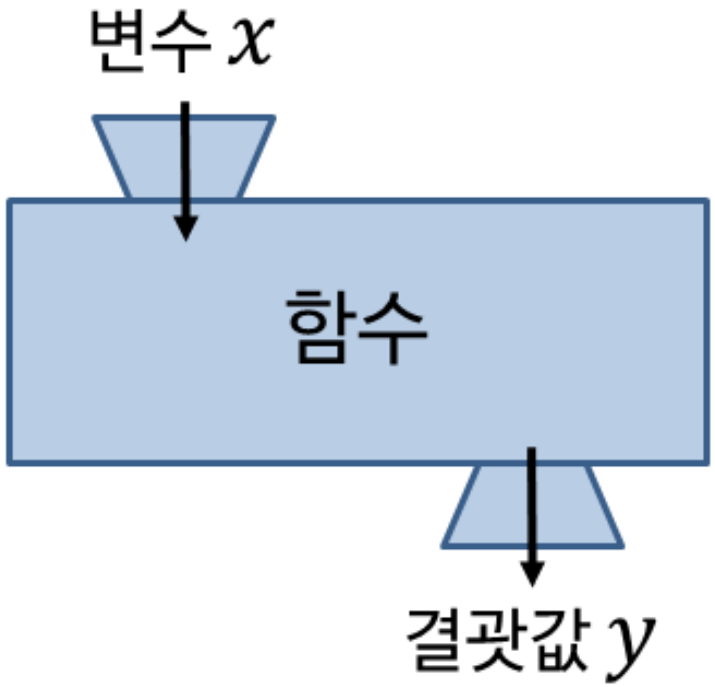
```
>>> a = {'a': 1, 'b': 2, 'c': 3}
>>> for key in a.keys():
...     print(a[key])
...
1
2
3
```

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

함수

함수란?

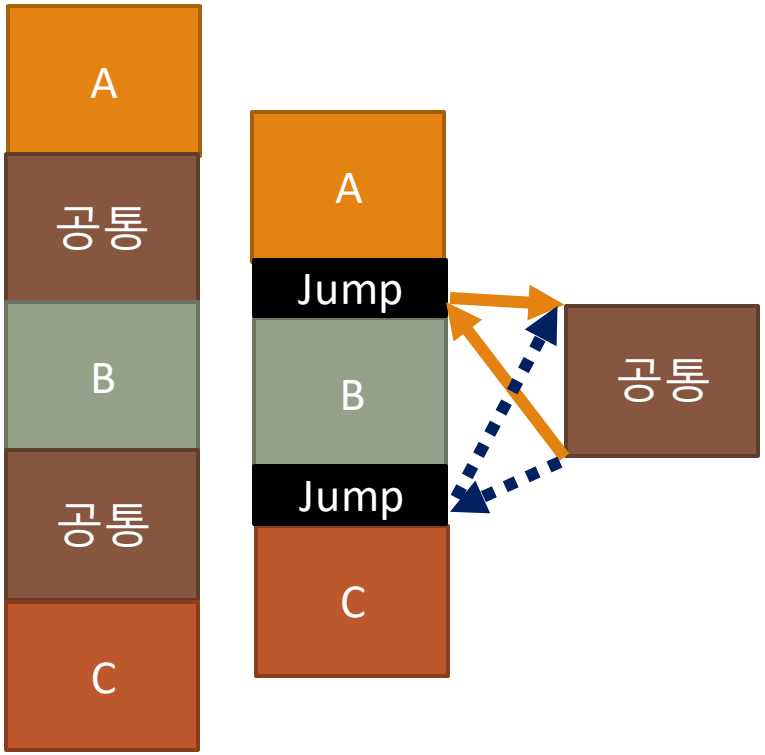
- 어떠한 값(x)를 집어넣었을 때, 결과값(y)가 출력되는 것



함수

함수를 사용하는 이유

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



함수

함수를 사용하는 이유

- 반복되는 내용을 재사용하기 위해 사용한다.

```
11  num = int(input("뽑으실 물품을 골라주십시오 : "))
12  if num < 1 or num > 4:
13      print("물품 번호를 잘못 입력하셨습니다.")
14      print("돈을 반환합니다.  [%d원] 짤랑 " % money)
15  elif num == 4:
16      print("돈을 반환합니다.  [%d원] 짤랑 " % money)
17
18  elif num == 1:
19      if money < 550:
20          print("금액이 부족합니다.")
21          print("돈을 반환합니다.  [%d원] 짤랑 " % money)
22      else:
23          print("(오라떼 사과)이/가 나왔습니다.")
24          print("거스름돈 %d원" % (money - 550))
25
26  elif num == 2:
27      if money < 800:
28          print("금액이 부족합니다.")
29          print("돈을 반환합니다.  [%d원] 짤랑 " % money)
30      else:
31          print("(갈아만든 배)이/가 나왔습니다.")
32          print("거스름돈 %d원" % (money - 800))
33
34  elif num == 3:
35      if money < 1000:
36          print("금액이 부족합니다.")
37          print("돈을 반환합니다.  [%d원] 짤랑 " % money)
38      else:
39          print("(박카스F)이/가 나왔습니다.")
40          print("거스름돈 %d원" % (money - 1000))
```

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

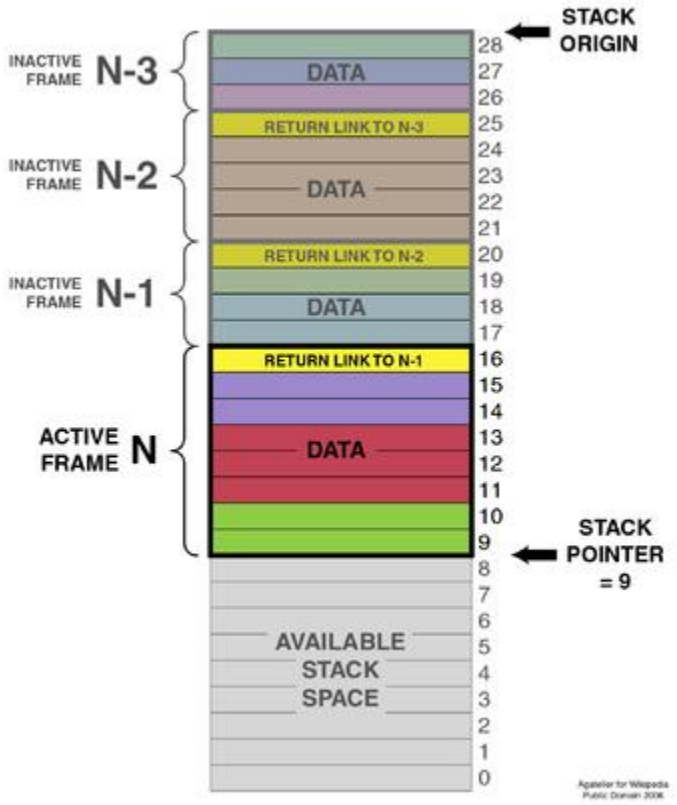
- 이 방식은 사람이 명령전체를 기억해야 되고, 변경 시 전체를 변경해야 한다.
- 돌아갈 목적지를 기억하는 메모리를 만들었다.

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

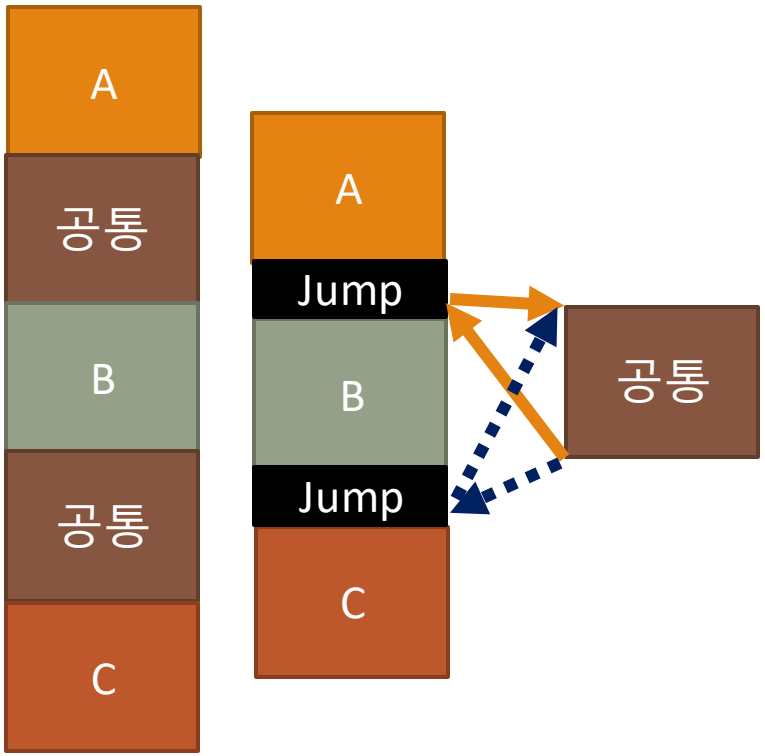
- 여러 함수를 연속적으로 호출 시,
  돌아갈 목적지 메모리의 내용이 사라진다.
- 스택(Stack)이 등장



함수

함수의 호출과 반환

- 호출(call): 함수를 사용하기 위해 부르는 행위
- 호출자(Caller): 함수를 부르는 코드(Jump)
- 반환(Return): 함수가 호출자에게 결과를 되돌려 줌



함수

함수의 구조

```
def 함수명(매개변수1, 매개변수2):
    수행문1
    수행문2
    return 반환값
```

함수

함수의 구조

- 매개변수(입력값, 입력인수, 함수인수 등)
  - 매개: 중간에서 둘 사이의 관계를 맺다
  - 함수를 호출할 때, 필요한 값을 받아오는 변수
  - 호출자는 반드시 매개변수의 개수만큼 값을 입력해야 한다.

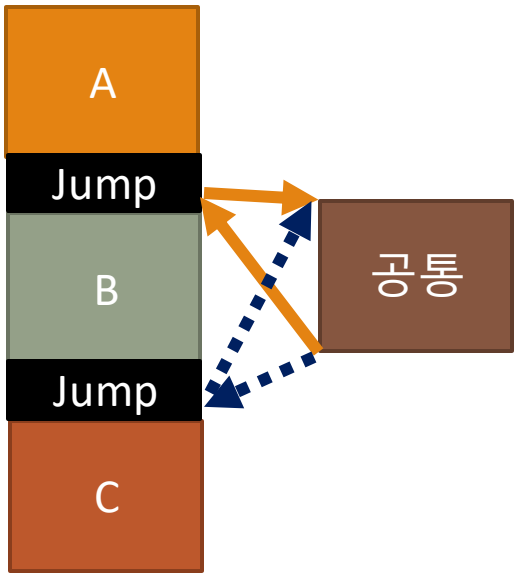
```
>>> def mul(a, b):
...     return a * b
...
>>> a = mul(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: mul() missing 1 required positional argument: 'b'
```

함수

매개변수(입력인수, 인자)

- 함수 밖의 변수명과는 관련이 없다.
- 매개변수는 함수가 끝나면 메모리에서 제거된다.

```
1  # 내부에 있는 변수명임으로, 바꿔도 상관 없다.
2  def test(value_list):
3      for value in value_list:
4          print(value_list)
5
6
7  value_list = [1, 2, 3]
8  test(value_list)
```



함수

함수의 구조

- 반환값(출력값, 리턴값, 돌려주는 값 등)
  - 함수가 필요한 작업을 완료하고 반환하는 값

함수

함수의 형태

- 함수는 매개변수가 없거나 반환값이 없을 수 있다.
  - 일반적인 함수

```
>>> def mul(a, b):
...     return a * b
...
>>> a = mul(2, 3)
>>> print(a)
6
```

  - 매개변수가 없을 때

```
>>> def my_title():
...     return "Python is easy!"
...
>>> a = my_title()
>>> print(a)
Python is easy!
```

함수

함수의 형태

- 함수는 매개변수가 없거나 반환값이 없을 수 있다.
  - 반환값이 없을 때

```
>>> def mul_print(a, b):
...     print("%d * %d = %d" % (a, b, a*b))
...
>>> mul_print(2, 3)
2 * 3 = 6
```

  - 둘 다 없을 때

```
>>> def my_print():
...     print("Python is easy!")
...
>>> my_print()
Python is easy!
```

함수

반환값이 없다?

- return이 없으면 None을 반환
- 객체가 없다는 것을 뜻함
- Bool에서는 False

```
>>> a = my_print()
Python is easy!
>>> a
>>> type(a)
<class 'NoneType'>
```

함수

매개변수(입력인수, 인자)

- 위치 매개변수(positional arguments)사용
  - 일반적으로 사용하는 매개변수
  - 순서대로 값을 받는다.
  - 각 위치를 사용자가 알아야 한다. (IDE가 도와주기도 한다.)

```
>>> def str_test(first, second):
...     print(first, second)
...
>>> str_test("Hello", "world")
Hello world
>>> str_test("world", "Hello")
world Hello
```

함수

매개변수(입력인수, 인자)

- 키워드 매개변수(keyword argument)
  - 위치에 대한 혼동없이, 매개변수의 이름으로 값을 넣는다.
  - 키워드를 입력하지 않는다면, 당연히 위치매개변수와 같은 방법으로 사용가능하다

```
>>> def str_test(first, second):
...     print(first, second)
...
>>> str_test(first="Hello", second="world")
Hello world
>>> str_test(second="world", first="Hello")
Hello world
>>> str_test("Hello", "world")
Hello world
```

함수

매개변수(입력인수, 인자)

- 기본값 매개변수(default arguent value)로도 사용되어, 값을 넣어주지 않아도, 기본값이 들어간다.

```
>>> def str_test(first="HI", second="friends"):
...     print(first, second)
...
>>> str_test()
HI friends
>>> str_test("hello")
hello friends
>>> str_test(second="world")
HI world
```

함수

매개변수(입력인수, 인자)

- 매개변수 일부만 기본값 매개변수 사용하기
  - 기본값이 있는 매개변수를 뒤에 놓는다.

```
>>> def example(first, second=2, third):
...     pass
```

  - example(12, 3) => 3을 어디에 대입?

  - 이런 함수를 만들면 다음 오류를 발생

```
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

함수

가변 매개변수

- 튜플 가변 매개변수(*),
  - 위치 매개변수를 튜플을 기반으로 하여 가변으로 묶어준다.
  - 관용적으로 *args(arguments: 인수, 매개변수)를 사용

```
>>> def sum_int(*args):
...     print(args)
...     temp = 0
...     for num in args:
...         temp += num
...     print(temp)
...
>>> sum_int(1, 2, 3)
(1, 2, 3)
6
>>> sum_int(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5)
15
```

함수

가변 매개변수

- 딕셔너리 가변 매개변수(**),
  - 키워드 매개변수를 매개변수 이름은 키, 매개변수 값은 값으로 하는 딕셔너리를 만든다.
  - 관용적으로 **kwargs(keyword arguments)를 사용한다.

```
>>> def kwargs_print(**kwargs):
...     print(kwargs)
...
>>> kwargs_print(this="this", be="is", sparta="sparta")
{'this': 'this', 'be': 'is', 'sparta': 'sparta'}
```

함수

키워드 전용 매개변수

- 키워드 매개변수로만 인자를 전달할 수 있다.
- 위치 매개변수로는 사용하지 못한다.

```
SyntaxError: invalid syntax
>>> def example(num, *args, kwd):
...     print(num)
...     print(*args)
...     print(kwd)
...
>>> example(1, 3, 4, 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: example() missing 1 required keyword-only argument: 'kwd'
>>> example(1, 2, 3, 4, kwd="kwd")
1
2 3 4
kwd
```

함수

키워드 전용 매개변수

- 튜블 기반 가변 매개변수를 사용하지 않을 때

```
>>> def example(pos, *, num):
...     print(pos)
...     print(num)
...
>>> example(1, 23)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: example() takes 1 positional argument but 2 were given
>>> example(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: example() missing 1 required keyword-only argument: 'num'
>>> example(num=23)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: example() missing 1 required positional argument: 'pos'
>>> example(1, num=23)
1
23
```

함수

키워드 전용 매개변수

- 튜플 기반 가변 매개변수를 사용하지 않을 때

```
>>> def example(*, num):
...     print(num)
...
>>> example(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: example() takes 0 positional arguments but 1 was given
>>> example(num=1)
1
```

함수

위치, 키워드 전용, 가변 매개변수의 혼합

- 위치 매개변수, 튜플 가변 매개변수,
  키워드 전용 매개변수, 딕셔너리 가변 매개변수 순서

함수

```
1   def example(pos, *args, kwd=None, **kwargs):
2       print("pos: %s" % pos)
3       if args:
4           for value in args:
5               print("args: %s" % value)
6       if kwd is not None:
7           print("kwd: %s" % kwd)
8       if kwargs is not None:
9           for key, value in kwargs.items():
10              print("kwargs, key: %s, value: %s" % (key, value))
11
12
13  example(1)
14  example(1, 3, 4, 5)
15  example(1, kwd=23)
16  example(pos=1, kwd=23, test=4, this="this")
17  mydic = {'pos': 1, 'kwd': "kwd", 'this': "this"}
18  example(**mydic)
```

```
pos: 1
pos: 1
args: 3
args: 4
args: 5
pos: 1
kwd: 23
pos: 1
kwd: 23
kwargs, key: test, value: 4
kwargs, key: this, value: this
pos: 1
kwd: kwd
kwargs, key: this, value: this
```

함수

함수의 결과값은 하나

- 두 개의 값도 반환이 잘 된다?

```
>>> def two_return():
...     return 1, 2
...
>>> a, b = two_return()
>>> a
1
>>> b
2
```

함수

함수의 결과값은 하나

- 튜플로 반환되기 때문에 가능
- 반환값은 하나이다.

```
>>> a = two_return()
>>> a
(1, 2)
```

자료형-튜플의 패킹과 언패킹

튜플 언패킹(tuple unpacking)

- 각 요소를 여러 개의 변수에 할당하는 것
- 이를 통해 두 개의 변수를 쉽게 바꿀 수 있다.
- 이를 통해 함수의 return시 여러 값을 넘길 수 있다.

```
>>> a = 1, 2, 3
>>> one, two, three = a
>>> one
1
>>> two
2
>>> three
3
```

함수

docstring(document string)

- 파이썬의 철학- 가독성은 중요하다(readability counts)
- 함수 정의 문서 (모듈이나 패키지에서도 사용)
- 함수 시작부분에 붙인다.

```
def coffee_list(coffeefile):
    """
    커피 목록에 대한 파일을 리스트로 반환한다

    파일에 있는 이름, 가격, 수량을 2차원 리스트로 반환한다.
    파일이 비어있으면 빈 리스트를 반환한다.

    :param coffeefile: 커피 filedescriptor
    :return coffeelist: 커피 리스트, 2차원이다.

    or

    Args:
        coffeefile: 커피 filedescriptor
    Returns:
        coffeelist: 커피 리스트, 2차원이다.
    """
```

함수

docstring(document string)

- 첫 줄은 함수가 수행하는 일을 한 문장 설명
- 다음 문단은 함수의 특별한 동작과 인수 및 반환값, 예외 에 대한 설명

```
def coffee_list(coffeefile):
    """
    커피 목록에 대한 파일을 리스트로 반환한다

    파일에 있는 이름, 가격, 수량을 2차원 리스트로 반환한다.
    파일이 비어있으면 빈 리스트를 반환한다.

    :param coffeefile: 커피 filedescriptor
    :return coffeelist: 커피 리스트, 2차원이다.

    or

    Args:
        coffeefile: 커피 filedescriptor
    Returns:
        coffeelist: 커피 리스트, 2차원이다.
    """
```

함수

docstring(document string)

- 인수 없이 간단한 값 반환 시 한 줄 설명이 좋다.
- 반환이 없으면 return은 생략하는게 좋다.
- docstring을 작성했으면 계속 업데이트 하자.

함수

docstring(document string)

- help()로 호출할 수 있다.

```
>>> import docstr
>>> help(docstr.coffee_list)
```

```
Help on function coffee_list in module docstr:

coffee_list(coffeefile)
    커피 목록에 대한 파일을 리스트로 반환한다

    파일에 있는 이름, 가격, 수량을 2차원 리스트로 반환한다.
    파일이 비어있으면 빈 리스트를 반환한다.

    :param coffeefile: 커피 filedescriptor
    :return coffeelist: 커피 리스트, 2차원이다.

    or

    Args:
        coffeefile: 커피 filedescriptor
    Returns:
        coffeelist: 커피 리스트, 2차원이다.
(END)
```

함수

docstring(document string)

- help()로 호출할 수 있다.
- 구름 IDE에서는 less나 more로 출력함으로, q를 입력하면 종료가능하다.
- pycharm IDE에서는 커서를 놓아둘 때, 도움말로 나타난다.

특별한 이름

-와 __사용

- 두 언더스코어(__)로 시작하고 끝나는 변수는 파이썬 내의 사용을 위해 예약되어 있다.
- 함수의 이름: function.__name__
- 함수의 docstring: function.__doc__

특별한 이름

-와 __사용

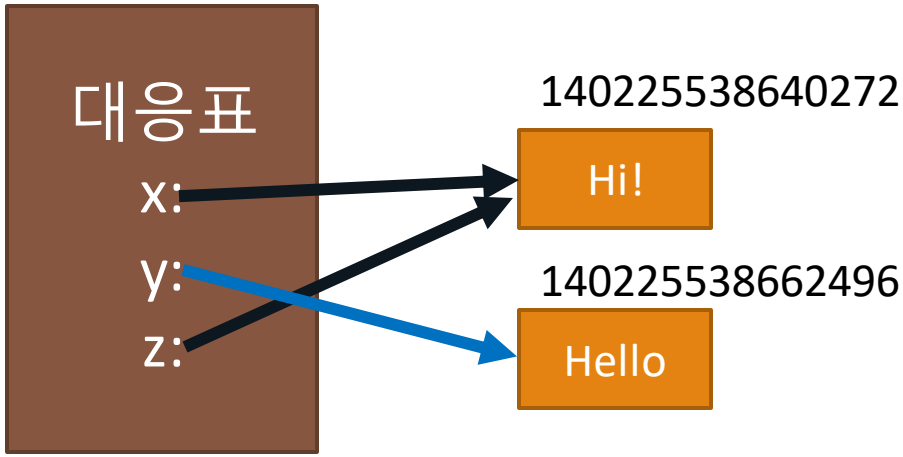
- 두 언더스코어(__)로 시작하고 끝나는 함수
- 마술 메서드(maginc method), 특별 메서드라고 한다.
- 스티브홀던
  - 던더(double under->dunder) 메서드
- 파이썬에서 미리 정의된 함수로 이를 통해 사용자 객체를 만들 때, 기본적인 객체연산이 가능하도록 한다.
- ex) __getitem__()

함수-네임스페이스와 스코프

이름

- 변수에서 보았듯이,
  이름이 있기 전에는 번호를 부여했다.
- 대응표를 만들어 이름을 붙인다.

```
>>> x = "Hi!"
>>> id(x)
140225538640272
>>> y = "Hello"
>>> id(y)
140225538662496
>>> z = x
>>> id(z)
140225538640272
```



함수-네임스페이스와 스코프

이름 충돌

- 초기 프로그래밍 언어에서는 대응표가 전체 공유되었다.
- 다음 루프는 10번 실행 후에 끝날까?

```
i = 0
while i < 10:
    short()
    print("처리중")
    i++
```

함수-네임스페이스와 스코프

이름 충돌

- 초기 프로그래밍 언어에서는 대응표가 전체 공유
- 만일 short()내에서 i를 사용한다면?

```
i = 0
while i < 10:
    short()
    print("처리중")
    i++
```

함수-네임스페이스와 스코프

충돌 피하기

- 긴 변수명을 사용
  - 프로그램이 커지면 힘들다.
  - 변수명을 만드는데 시간이 오래 걸린다.
- scope(스코프, 범위)의 탄생
  - 이름의 유효범위를 좁혀 관리가 편하게 한다.
  - 스코프가 다르면 같은 이름이 다른 개체를 가리키게 한다.

함수-네임스페이스와 스코프

스코프의 종류

- 동적 스코프
  - 함수 입구에 원래 값을 기록해두고, 출구에서 원래의 값으로 되돌린다.
- 정적 스코프
  - 대응표를 함수의 호출마다 다르게 둔다.
  - 파이썬에서 사용하는 방법

- 스코프에 대한 더 자세한 내용은 코딩을 지탱하는 기술 7장을 참고한다.

함수-네임스페이스와 스코프

네임스페이스와 스코프

- 네임스페이스(Name space, 이름공간)
  파이썬내의 대응표를 이렇게 부른다.

- 변수의 스코프(scope, 범위)
  변수의 이름으로 그 변수가 가리키는 값을 찾을 수 있는 영역의 범위

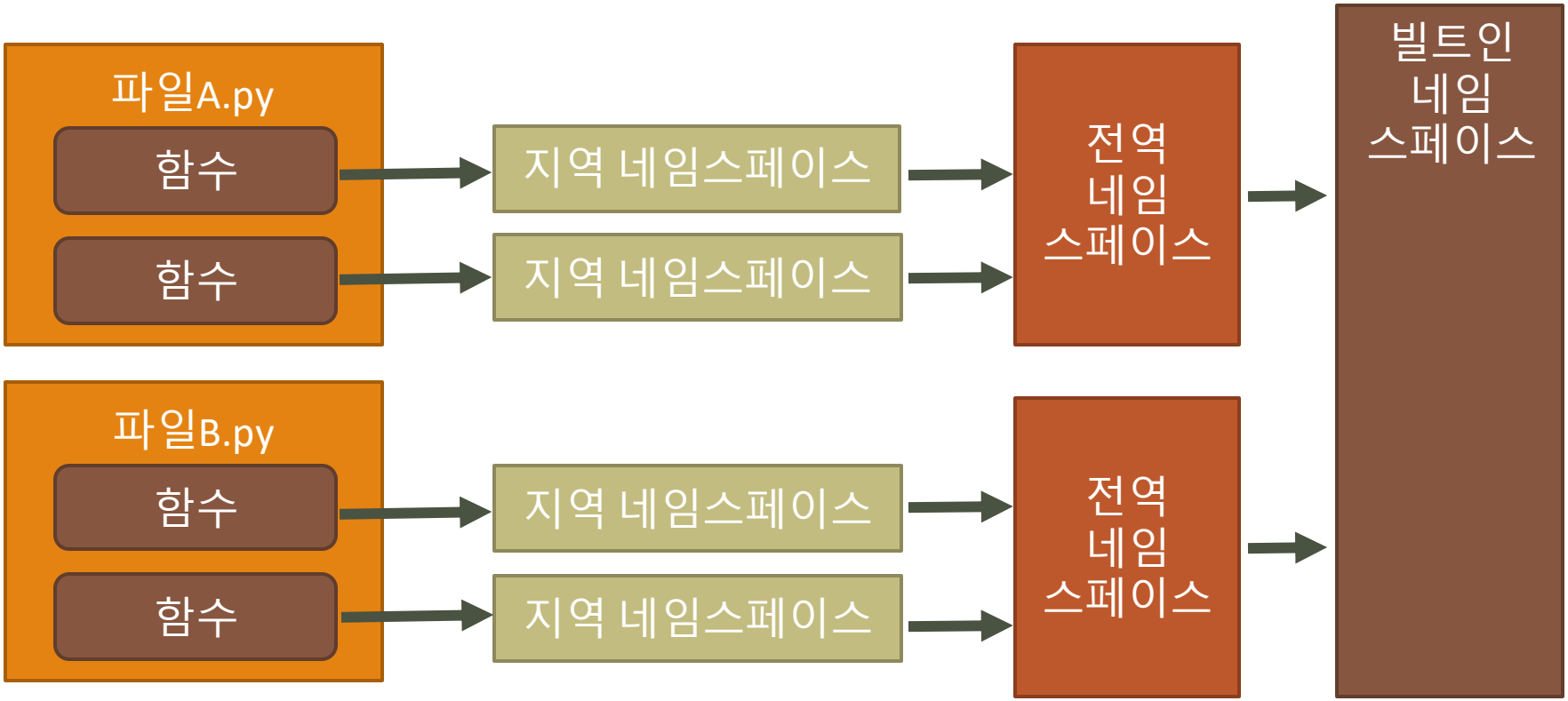
함수-네임스페이스와 스코프

Python의 네임스페이스

- 지역 네임스페이스(local namespace)
  - 함수 및 메서드별로 존재
  - 함수 내의 지역변수가 소속
- 전역 네임스페이스(global namespace)
  - 모듈별로 존재, 모듈 전체에 통용되는 이름을 사용
- 빌트인 네임스페이스(built-in namespace)
  - 기본 내장 함수 및 기본 예외들의 이름을 저장
  - str()등

함수-네임스페이스와 스코프

Python의 네임스페이스는 다음과 같다.



함수-네임스페이스와 스코프

전역변수와 지역변수

- 전역변수
  - 모듈(파일)의 최상위에서 선언한 이름
  - globals()를 통해 전역 네임스페이스의 사본 확인이 가능

```
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins'
```

- 지역변수
  - 함수나 메소드 단위로 생성
  - 함수에 진입하는 시점에 생성
  - locals()함수를 호출하면 된다.

```
>>> def func():
...     m = 1
...     n = 2
...     print(locals())
...
>>> func()
{'n': 2, 'm': 1}
```

함수-네임스페이스와 스코프

전역변수와 지역변수

- 예시

```
>>> x = 1
>>> def vardefine(x):
...         x += 3
...
>>> print(x)
1
>>> vardefine(x)
>>> print(x)
1
```

함수-네임스페이스와 스코프

쉐도잉(shadowing)

- 네임 마스킹이라고도 한다.
- 특정 스코프 내에 선언된 이름이 외부 스코프와 중첩
  - 이 경우 내부 스코프에 있는 값을 사용
- 변수명 참조 시 다음과 같이 참조
  - 지역 네임스페이스
  - 없을 경우 전역 네임스페이스
  - 없을 경우 빌트인 네임스페이스
  - 최상단에도 없을 경우 NameError예외를 발생시킨다.

함수-네임스페이스와 스코프

쉐도잉(shadowing)

- 알아차리기 어려운 경우
- 쉐도잉은 읽는 경우에만 적용
- 'num =' 즉, 할당을 하는 순간, 로컬 변수를 초기화하며 지역 네임스페이스에 등록한다

```
num = 3


def increse_num(step=2):
    num = num + step


increse_num()
print(num)
```

```
Traceback (most recent call last):
  File "shadowing.py", line 8, in <module>
    increse_num()
  File "shadowing.py", line 5, in increse_num
    num = num + step
UnboundLocalError: local variable 'num' referenced before assignment
```

함수-네임스페이스와 스코프

명시적으로 상위/전역 스코프 변수 사용

- 파이썬 철학: 명확한 것이 함축적인 것보다 낫다.
- global
  - 전역 변수를 사용한다는 선언

```
>>> vardefine()
>>> x = 1
>>> def vardefine():
...     global x
...     x += 3
...
>>> vardefine()
>>> print(x)
4
```

함수-네임스페이스와 스코프

명시적으로 상위/전역 스코프 변수 사용

- global
  - 웬만하면 쓰지 않는다.
  - 스코프를 만든 이유-이름 충돌을 막기 위해서
  - 코드가 복잡해지면, 변수의 값이 어디서 바뀌는지 알 수 없다.

함수-네임스페이스와 스코프

명시적으로 상위/전역 스코프 변수 사용

- global
  - return을 통해 함수를 호출한 곳에서 값을 갱신하도록 하는게 좋다.

```
>>> def vardefine(x):
...     x += 3
...     return x
...
>>> x = 1
>>> x = vardefine(x)
>>> print(x)
4
```

함수-네임스페이스와 스코프

명시적으로 상위/전역 스코프 변수 사용

- nonlocal
- 함수의 중첩과 관련이 있어 심화에서 다룬다.
- 기초적인 프로그래밍에서는 사용할 가능성이 적다.

함수-네임스페이스와 스코프

실수인가?

- 다음은 어떻게 실행될까?

```
1   x = 3
2
3
4   def my_func():
5       print(x)
6       x += 1
7       print(x)
8
9
10  my_func()
```

함수-네임스페이스와 스코프

실수인가?

- 인터프리터 언어-한 줄씩 해석한다?
- 한 줄씩 해석하는 것은 모듈을 로드할 때만 이다.
- 소스코드를 받으면 모든 해석이 이뤄진다.
- 해석을 한 이후 소스코드는 컴파일된 바이트 코드 (기계어에 가까운 코드)로 번역된다
- x의 네임스페이스는 첫번째 print(x)를 하기 전에 지역으로 정해진다.

```
1   x = 3
2
3
4   def my_func():
5       print(x)
6       x += 1
7       print(x)
8
9
10  my_func()
```

```
Traceback (most recent call last):
  File "local_namespace_error.py", line 10, in <module>
    my_func()
  File "local_namespace_error.py", line 5, in my_func
    print(x)
UnboundLocalError: local variable 'x' referenced before assignment
```

함수-네임스페이스와 스코프

실수인가?

- 대화형 인터프리터에서도 마찬가지이다.

```
>>> x = 3
>>>
>>>
>>> def my_func():
...     print(x)
...     x += 1
...     print(x)
...
>>> my_func()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in my_func
UnboundLocalError: local variable 'x' referenced before assignment
```

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-자판기4

자판기(vending_machine.py)

- 배운 내용을 바탕으로 새로운 기능을 추가한다.
- 이전의 프로그램에 추가한다.
- 1. 물품의 개수를 변경할 수 있는 admin모드를 추가
- 1.1 admin과 같은 문자열을 물품을 고르는 입력창에 입력하면 admin모드에 접근할 수 있다.
- 1.2 물품의 개수는 일반 사용자가 볼 수 없다.
- 단, 정수와 접근문자열이 아니면 다시 입력받는다.

기본과제-자판기4

자판기(vending_machine.py)

- 2. admin모드에 접속하면 다음의 작업을 선택할 수 있도록 리스트를 출력한다.
    1. 물품목록과 개수 출력
    2. 물품개수 변경
    3. 종료
- 3. 물품목록과 개수 출력을 누르면 물품목록과 개수를 출력하고 선택창을 다시 출력한다.
- 4. 물품개수 변경을 클릭하면 물품목록과 개수를 다시 출력하고, 물품을 선택했을 때, 물품 개수를 변경할 수 있게 한다.

기본과제-자판기4

자판기(vending_machine.py)

- 5. 물품 개수를 변경하면 admin모드 첫 화면으로 되돌아간다.
- 6. Admin모드에서 종료를 선택하면, 맨 처음 자판기 화면으로 돌아간다
- 7. 물품을 뽑았을 때, 물품 개수가 부족하면 "물품이 부족합니다."를 출력하고 처음으로 되돌아간다.
- 8. 일반 모드에서 '종료'를 출력하여 종료를 선택할 때 종료하도록 한다.
- 단, 이전 종료방식(음수 선택)은 삭제한다.

기본과제-자판기4

자판기(vending_machine.py)

- 예시

```
1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
6. 종료
현재까지 넣은 돈은 400원입니다.
뽑을 물품을 골라주세요: 1
물품 개수가 부족합니다. 관리자를 불러주세요.

1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
6. 종료
현재까지 넣은 돈은 400원입니다.
뽑을 물품을 골라주세요: admin
```

```
1. 물품출력
2. 개수추가
3. 종료
원하는 작업을 선택해주세요: 1

1. 블랙커피(100원) 개수: 0
2. 밀크커피(150원) 개수: 1
3. 고급커피(200원) 개수: 1

1. 물품출력
2. 개수추가
3. 종료
원하는 작업을 선택해주세요: 2

1. 블랙커피(100원) 개수: 0
2. 밀크커피(150원) 개수: 1
3. 고급커피(200원) 개수: 1
개수를 추가할 물품을 선택하세요: 1
추가할 개수를 입력해주세요: 3
```

기본과제-자판기4

자판기(vending_machine.py)

- 예시

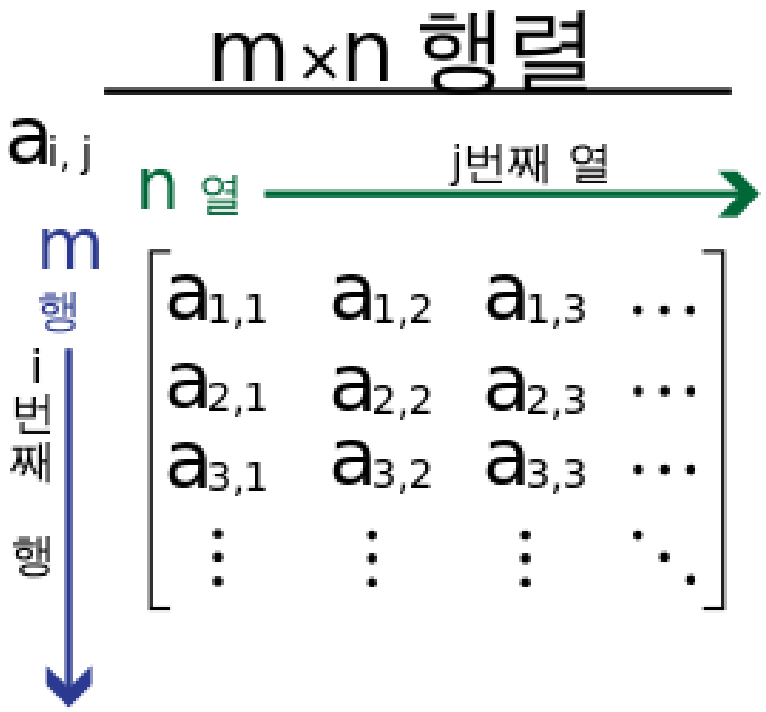
```
1. 물품출력
2. 개수추가
3. 종료
원하는 작업을 선택해주세요: 1

1. 블랙커피(100원) 개수: 3
2. 밀크커피(150원) 개수: 1
3. 고급커피(200원) 개수: 1

1. 물품출력
2. 개수추가
3. 종료
원하는 작업을 선택해주세요: 3
```

```
자판기 모드로 돌아갑니다.
1. 블랙커피(100원)
2. 밀크커피(150원)
3. 고급커피(200원)
4. 돈 입력
5. 거스름돈
6. 종료
현재까지 넣은 돈은 400원입니다.
뽑을 물품을 골라주세요: 1
블랙커피이/가 나왔습니다.
```

심화과제-달팽이

달팽이 그리기(snail.py)

- 행렬이란, 수나 기호, 수직등을 네모꼴로 배열한 것



심화과제-달팽이

달팽이 그리기(snail.py)

- 연속되는 숫자를 행렬을 통해 달팽이 모양으로 배열시킨다.
- 다음과 같은 n*m배열 구조를 출력해보자.
- 입력이 행: 3, 열: 4인 경우 다음과 같이 출력한다.
- 1 2 3 4
  10 11 12 5
  9 8 7 6

http://koistudy.net/?mid=prob_page&NO=1735&SEARCH=0

심화과제-달팽이

달팽이 그리기(snail.py)

- 입력이 행: 4, 열: 5인 경우는 다음과 같이 출력한다.
- 1 2 3 4 5
  14 15 16 17 6
  13 20 19 18 7
  12 11 10 9 8
- 단, 입력받는 수는 n, m <= 30이다.
- 음수를 받으면 프로그램을 종료한다.

http://koistudy.net/?mid=prob_page&NO=1735&SEARCH=0

심화과제-달팽이

달팽이 그리기(snail.py)

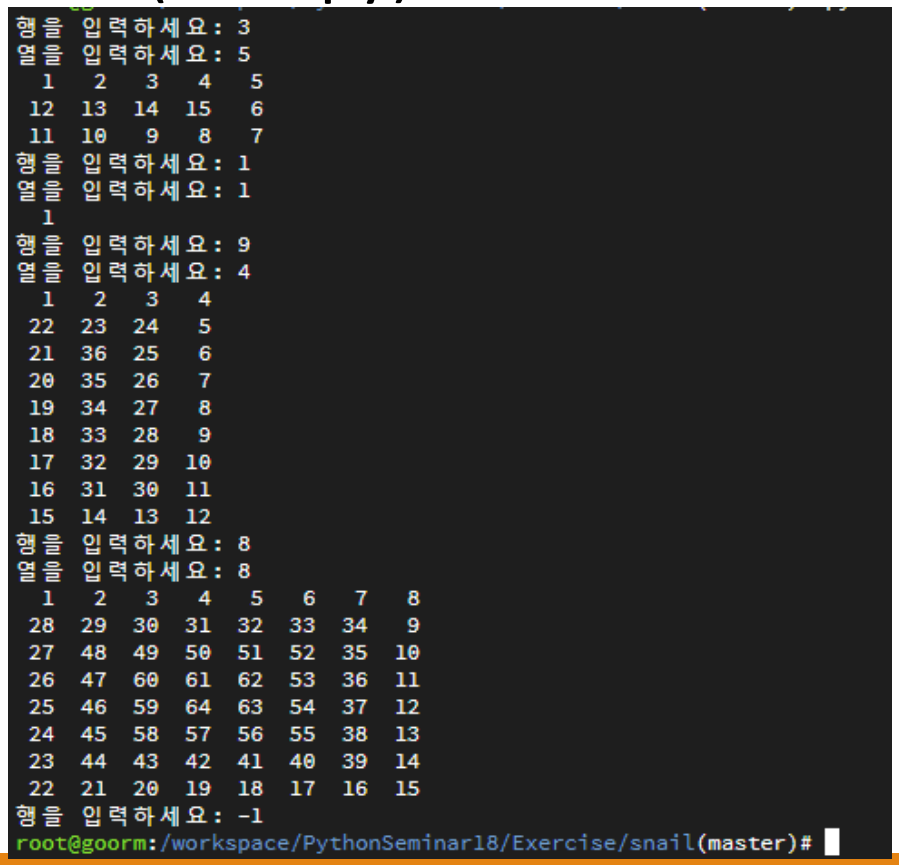
- ※ 이중 리스트를 배열(행렬)로 사용할 수 있다.
- ※ 이중 리스트의 접근은 c언어의 배열과 같다.

http://koistudy.net/?mid=prob_page&NO=1735&SEARCH=0

심화과제-달팽이

달팽이 그리기(snail.py)

- 예시



http://koistudy.net/?mid=prob_page&NO=1735&SEARCH=0