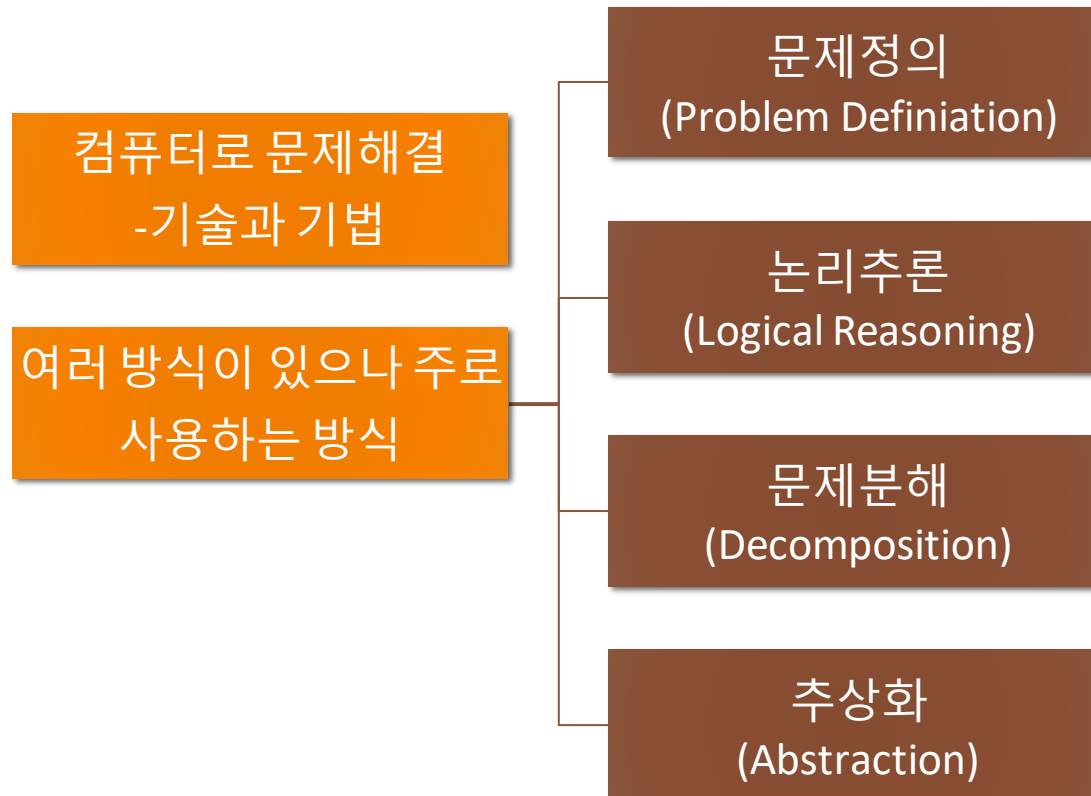


10. 논리추론, 조건문₩ 자료형(리스트, 튜플, bool)

2018.12

일병 김재형

문제해결



문제해결2-논리추론

문제정의

- 어떠한 것을 개발하고 어떤 기능이 있는지 정의
- 요구사항을 토대로 개발 방향을 정한다.

문제해결2-논리추론

논리적 추론의 두 가지 방법론

- 기능적 요구사항을 원인-결과 관계로 분석
- 연역적 추리 방법을 응용함

문제해결2-논리추론

원인-결과 관계

- 논리적인 조건(원인)을 통해 작업(결과)을 수행하도록 구성
- if(원인) then (결과)의 방식으로 프로그래밍

| 원인 | 결과 |
|----------------|--------------------------|
| 아이디와 비밀번호가 입력됨 | 아이디와 비밀번호가 유효한지 확인하고 로그인 |
| 자판기에 돈을 입력 | 넣은 돈의 값이 증가함 |
| 새 쪽지가 옴 | 사용자에게 알람을 띄움 |
| 입대영장이 날라옴 | 군대에 입대함 |

문제해결2-논리추론

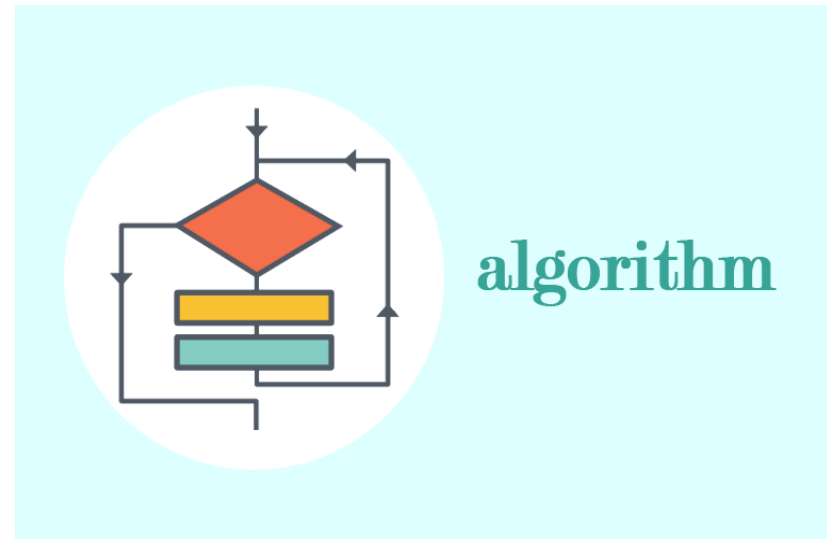
연역적 추리

- 특정 상황에 일반적인 규칙을 적용하여 문제 해결
- 직각 삼각형의 빗변의 길이(c) (특정상황)를 구하기 위해 피타고라스 정리(일반규칙)을 사용

문제해결2-논리추론

순차화

- 문제를 해결하기 위해 작업들의 선후관계를 기술
- 알고리즘: 어떤 작업을 수행하기 위해 입력을 받아 원하는 출력을 만들어내는 과정
- 정확한 순서를 가져야 유효한 알고리즘이 된다.



```
>>> priceWithTax = 0
>>> itemCost = 100
>>> priceWithTax = itemCost + itemCost*0.55
>>> priceWithTax
155.0
```

```
>>> itemCost = 100
>>> priceWithTax = itemCost + itemCost*0.55
>>> priceWithTax=0
>>> priceWithTax
0
```

문제해결2-논리추론

패턴

- 문제를 해결하기 위한 규칙은 '패턴'의 형태로 구성되기도 함
- 두 변수의 내용을 교환하는 문제

```
temp ← varA  
varA ← varB  
varB ← temp
```

```
temp ← myDog  
myDog ← yourDog  
yourDog ← temp
```

그림 4.5 교환 패턴

```
>>> myDog="JaeHyeong"  
>>> yourDog="YoungChun"  
>>> myDog, yourDog = yourDog, myDog  
>>> myDog  
'YoungChun'  
>>> yourDog  
'JaeHyeong'
```


문제해결2-논리추론

반복 패턴

- 특정 작업을 반복적으로 실행할 때 사용하는 패턴
- 컴퓨터가 가장 잘하는 것이 반복 작업임으로 가장 흔하게 발생한다.
- ex) 1-1000000출력



```
>>> for i in range(100):  
...     print(i)  
...  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

문제해결2-논리추론

귀납추론

—개별 사례에서 일반적 규칙을 도출

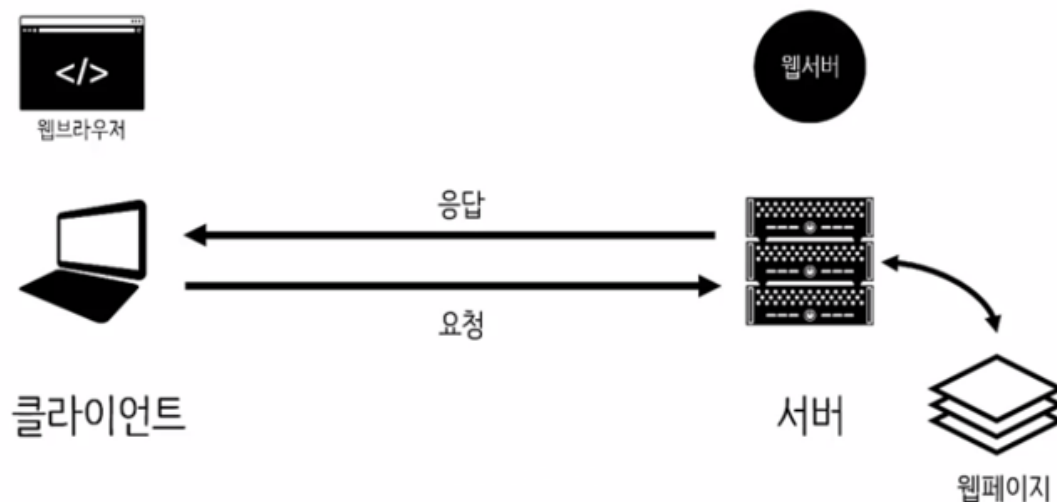
—Ex) 3단 소프트웨어 구조

성공적인 소프트웨어 솔루션들의 구조를 많이 관찰하여 특수한 규칙을 도출한 구조



문제해결2-논리추론

- ex) 3단 소프트웨어 구조 패턴
 - 웹서버는 다음의 3단 소프트웨어 구조 패턴으로 구현
 - 1. 사용자 통신하는 소프트웨어
 - 2. 데이터를 읽고 저장하는 소프트웨어
 - 3. 사용자 입력과 저장된 데이터를 기반으로 계산하는 소프트웨어



자료형-리스트

리스트(list, [])

- 데이터의 목록을 다루는 자료형
- 데이터를 모아 하나로 다룸 (명함집)
- 만일 100개의 값을 저장한다면, 변수 100개를 만들기 어렵다.
- 가변형이다.

※ C언어나 C++의 배열과 유사하나, 차이점이 있고, 자료구조의 리스트를 생각하는 것이 더 좋다.

```
a1 = 10
a2 = 20
# ... (생략)
a29 = 60
a30 = 40
```

자료형-리스트 생성

리스트 만들기

—리스트 변수명=[요소1, 요소2, 요소3, ...]

```
>>> even = [0, 2, 4, 6, 8, 10]
>>> even
[0, 2, 4, 6, 8, 10]
```

```
>>> a = []
>>> b = [1, 2, 3, 4]
>>> c = ['Python', 'is', 'easy']
>>> d = [1, 2, ['Python', 'is'], 3]
>>> d
[1, 2, ['Python', 'is'], 3]
```

자료형-리스트 생성

리스트의 요소(Element)

- 리스트 내부에 들어간 개별데이터를 요소라 한다.

```
>>> a = []  
>>> b = [1, 2, 3, 4]  
>>> c = ['Python', 'is', 'easy']  
>>> d = [1, 2, ['Python', 'is'], 3]  
>>> d  
[1, 2, ['Python', 'is'], 3]
```

- 비어 있는 리스트가 가능하다.
- 리스트가 리스트를 요소로 가질 수 있다.

자료형-리스트

리스트 \subset 시퀀스 자료형

- 연산
 - 연결
 - 반복
- 인덱스
- 슬라이싱

자료형-리스트 연산

연산

- 문자열과 같다.
- 연결하기(+)

```
>>> a = [1, 2, 3, 4]
>>> b = [2, 3, 5]
>>> a+b
[1, 2, 3, 4, 2, 3, 5]
```

- 반복하기(*)

```
>>> a = [1, 2, 3, 4]
>>> a*3
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```


자료형-리스트 인덱스

인덱스

—문자열과 같다.

```
>>> even
[0, 2, 4, 6, 8, 10]
>>> even = [0, 2, 4, 6, 8, 10]
>>> even[0]
0
>>> even[0]+even[2]
4
```

—마찬가지로 음수로도 인덱싱할 수 있다.

```
>>> even[-1]
10
```

자료형-리스트 인덱스

이중 리스트 인덱싱

—※ C언어의 배열과 유사

```
>>> a = [1, ['a', 'b', 'c'], 3]
>>> a[1]
['a', 'b', 'c']
>>> a[1][1]
'b'
```

삼중 리스트 인덱싱

```
>>> a = [1, ['a', 'b', ['Python', 'Easy']], 3]
>>> a[1][2][1]
'Easy'
```

자료형-리스트 슬라이싱

슬라이싱

—문자열과 같다.

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> b = a[1:3]
>>> c = a[4:]
>>> b
[2, 3]
>>> c
[5, 6]
```

자료형-리스트 슬라이싱

중첩된 리스트에서 슬라이싱

```
>>> a = [1, ['a', 'b', ['Python', 'Easy']], 3]
>>> a[1][0:2]
['a', 'b']
```

자료형-리스트 요소 변경

요소수정

—하나의 값 수정

```
>>> a = [1, 2, 3, 4]
>>> a[1] = 5
>>> a
[1, 5, 3, 4]
```

—연속된 범위의 값 수정

```
>>> a
[1, 5, 3, 4]
>>> a[1:3] = ['a', 'b']
>>> a
[1, 'a', 'b', 4]
```

자료형-리스트 요소 변경

요소수정

—하나의 값 vs. 연속된 범위의 값 리스트로 수정

```
>>> a = [1, 2, 3, 4]
>>> a[1] = ['a', 'b']
>>> a
[1, ['a', 'b'], 3, 4]
```

```
>>> a
[1, 5, 3, 4]
>>> a[1:3] = ['a', 'b']
>>> a
[1, 'a', 'b', 4]
```

자료형-리스트 요소 삭제

요소삭제

—빈 리스트 사용

```
>>> a = [1, 2, 3]
>>> a[1:2] = []
>>> a
[1, 3]
```

```
>>> a = [1, 2, 3]
>>> a[1] = []
>>> a
[1, [], 3]
```

—del함수 사용

```
>>> a = [1, 2, 3]
>>> del(a[1])
>>> a
[1, 3]
```

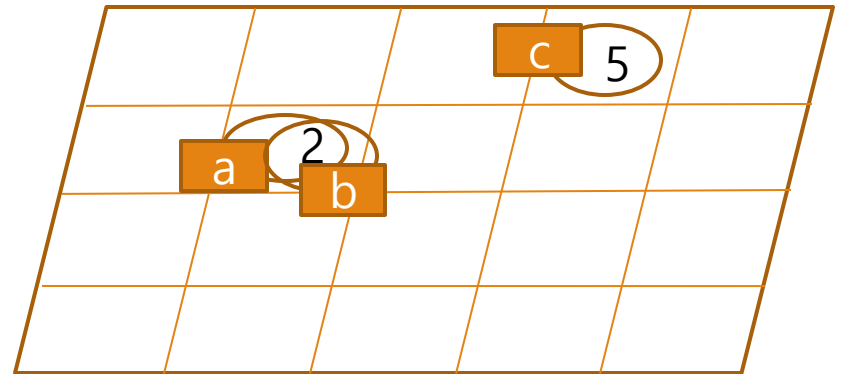
—메서드로 제거(remove, pop)

자료형-리스트 복사

단순 복제

—변수명만 다르다.

```
>>> a = [1, 2, [3, 4]]
>>> b = a
>>> id(a)
140168266856008
>>> id(b)
140168266856008
>>> b[0] = 0
>>> b
[0, 2, [3, 4]]
>>> a
[0, 2, [3, 4]]
```



자료형-리스트 복사

얕은 복사

- 외부 리스트만 별도로 생성

```
>>> a = [1, 2, [3, 4]]
>>> b = a[:]
>>> id(a)
140168266857032
>>> id(b)
140168266806920
>>> b[0] = 0
>>> a
[1, 2, [3, 4]]
>>> b
[0, 2, [3, 4]]
>>> b[2][0]=0
>>> b
[0, 2, [0, 4]]
>>> a
[1, 2, [0, 4]]
```

```
>>> import copy
>>> a = [1, 2, [3, 4]]
>>> b = copy.copy(a)
>>> id(a)
140168266855816
>>> id(b)
140168266856968
```

자료형-리스트 복사

깊은 복사

— 내부 내용도 복사

```
>>> import copy
>>> a = [1, 2, [3, 4]]
>>> b = copy.deepcopy(a)
>>> b
[1, 2, [3, 4]]
>>> b[0]=0
>>> b[2][0] = 0
>>> a
[1, 2, [3, 4]]
>>> b
[0, 2, [0, 4]]
```

자료형-리스트 메서드

리스트 요소 삽입(insert(첨자, 데이터))

—첨자의 위치에 새 요소 삽입

```
>>> a = [1, 2, 3, 4]
>>> a.insert(1, 5)
>>> a
[1, 5, 2, 3, 4]
```

자료형-리스트 메서드

리스트 요소 제거(remove(x))

- 리스트에서 처음 나온 x를 삭제한다.

```
>>> a = [1, 2, 3, 4, 3]
>>> a.remove(3)
>>> a
[1, 2, 4, 3]
>>> a.remove(3)
>>> a
[1, 2, 4]
>>>
>>> a.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

자료형-리스트 메서드

리스트 요소 추가(append(x))

—리스트의 맨 마지막에 x를 추가

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> a.append(['a', 'b'])
>>> a
[1, 2, 3, 4, ['a', 'b']]
```

자료형-리스트 메서드

리스트 확장(extend(x))

- 리스트의 맨 마지막에 리스트인 x를 추가
- +연산자와 같은 기능

```
>>> a = [1, 2, 3]
>>> a.extend([4])
>>> a
[1, 2, 3, 4]
>>> a.extend(['a', 'b'])
>>> a
[1, 2, 3, 4, 'a', 'b']
```

자료형-리스트 메서드

리스트 요소 끄집어내기(pop())

- 리스트의 마지막 요소를 뽑아내어 리스트에서 제거

```
>>> a = [1, 2, 3, 4, 5]
>>> a.pop()
5
>>> a
[1, 2, 3, 4]
>>> a.pop()
4
>>> a
[1, 2, 3]
```

자료형-리스트 메서드

리스트 요소 끄집어내기(pop(x))

- 리스트에서 x위치의 요소를 뽑아내고 리스트에서 제거

```
>>> a = [1, 2, 3, 4, 5]
>>> a.pop(2)
3
>>> a
[1, 2, 4, 5]
>>> a.pop(2)
4
>>> a
[1, 2, 5]
```


자료형-리스트 메서드

리스트 요소 끄집어내기(pop(x))

- append()와 반대의 기능
- 스택에서 사용하는 이름과 같은데, pop의 반대
가 push가 아닌 이유
 - 파이썬 창시자인 귀도가 1991년 초기에 append를 고안
 - 1997년 pop이 고안
 - 창시자도 더 적절하다고 생각하나, 같은 일을 하는 메소드를
다르게 구현하고 싶지 않아함
- 스택과 관련된 자세한 내용은 (심화)자료구조에서
수업한다.

자료형-리스트 메서드

리스트 정렬(sort())

- 리스트의 요소를 순서대로 정렬한다.

```
>>> a = [2, 4, 5, 1]
>>> a.sort()
>>> a
[1, 2, 4, 5]
>>> a = ['ad', 'a', 'aa']
>>> a.sort()
>>> a
['a', 'aa', 'ad']
>>>
>>> a = ["안녕 ", "간디 ", "하마 "]
>>> a.sort()
>>> a
['간디 ', '안녕 ', '하마 ']
```

자료형-리스트 메서드

리스트 정렬(sort())

- 키워드 매개변수 `reverse = True`를 사용하면 내림차순이 된다.

```
>>> a
['간디', '안녕', '하마']
>>> a.sort(reverse = True)
>>> a
['하마', '안녕', '간디']
```

자료형-리스트 메서드

리스트 뒤집기(reverse())

—리스트를 뒤집는다.

```
>>> a = [1, 5, 3, 4]
>>> a.reverse()
>>> a
[4, 3, 5, 1]
```

자료형-리스트 메서드

리스트 요소 세기(count())

- 입력한 데이터와 일치하는 요소가 몇 개 있는지 센다.

```
>>> a = [1, 10, 2, 3, 10]
>>> a.count(1)
1
>>> a.count(10)
2
>>> a.count(0)
0
```

자료형-리스트 메서드

리스트 요소 위치 반환(index())

- 매개변수로 입력한 데이터와 일치하는 첫 번째 요소의 첨자를 알려줌

```
>>> a = [1, 2, 1, 3, 4]
>>> a.index(1)
0
>>> a.index(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 5 is not in list
```

자료형-튜플()

튜플(tuple, ())

- 'N개의 요소로 된 집합'
- 리스트와 유사
- 불변형이어서 내부 값을 바꿀 수 없다.

자료형-튜플

튜플의 용도

- 불변형인 자료형임으로 변경이 불가능
- RGB값이나 위 경도 좌표와 같이 작은 곳에 사용
- 값이 항상 변하지 않기를 바랄 때 사용



자료형-튜플 생성

튜플의 생성

```
>>> a = ()
>>> a
()
>>> b = (1, 2, 3)
>>> b
(1, 2, 3)
>>> c = 1, 2, 3
>>> c
(1, 2, 3)
>>> d = (1, 2, (3, 4))
>>> d
(1, 2, (3, 4))
```

자료형-튜플 생성

요소가 하나인 튜플 생성

```
>>> a = 1
>>> type(a)
<class 'int'>
>>> a = 1,
>>> type(a)
<class 'tuple'>
>>> a
(1,)
>>> a = (1,)
>>> a
(1,)
>>> type(a)
<class 'tuple'>
```

자료형-튜플

튜플 \subset 시퀀스 자료형

- 연산
 - 연결
 - 반복
- 인덱스
- 슬라이싱

자료형-튜플 연산

연산

—연결하기

```
>>> a = (1, 2)
>>> b = (3, 4)
>>> a+b
(1, 2, 3, 4)
```

—반복하기

```
>>> a
(1, 2)
>>> a*3
(1, 2, 1, 2, 1, 2)
```

자료형-튜플 인덱싱 & 슬라이싱

인덱싱과 슬라이싱

```
>>> a = 1, 2, 3, 4, 5, 6
>>> a[1]
2
>>> a[2:4]
(3, 4)
```

```
>>> del a[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>> a[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

자료형-튜플의 패킹과 언패킹

튜플 패킹(tuple packing)

—여러가지 데이터를 튜플로 묶는 것

```
>>> a = 1, 2, 3
>>> a
(1, 2, 3)
```

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
```

자료형-튜플 메서드

튜플 요소 세기(count(x))

- 입력한 데이터와 일치하는 요소가 몇 개인지 센다.

```
>>> a = (1, 1, 1, 2, 2, 3)
>>> a.count(1)
3
>>> a.count(9)
0
```


자료형-튜플 메서드

튜플 요소 위치 반환(index())

- x와 일치하는 튜플 내 요소의 첨자를 알려준다.

```
>>> a = (1, 2, 3, 4)
>>> a.index(3)
2
>>> a.index('a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: tuple.index(x): x not in tuple
```

자료형-bool

bool(boolean, 불)

- 참과 거짓을 나타내는 형태
 - 불 자료형에는 True와 False만 존재한다.
- ※ 논리연산에서 쓰는 불 대수를 생각해도 된다.

```
>>> a = True
>>> a
True
>>> type(a)
<class 'bool'>
```

```
>>> a = False
>>> a
False
>>> type(a)
<class 'bool'>
```

자료형의 참/거짓

| 자료형 | 참 | 거짓 |
|----------|--------------|-------|
| 숫자 | 0이 아닌 숫자 | 0 |
| 문자열 | 비어있지 않은 문자열 | "" |
| 리스트 | 비어있지 않은 리스트 | [] |
| 튜플 | 비어있지 않은 튜플 | () |
| 딕셔너리 | 비어있지 않은 딕셔너리 | {} |
| 불 | True | False |
| Nonetype | | None |

자료형-자료형의 변환

자료형 변환

- list(반복 가능 객체(iterable object))
- tuple(반복 가능 객체(iterable object))
- bool(객체)
 - 값이 있으면 True이다.
- 반복 가능 객체란,
값을 차례대로 꺼낼 수 있는 객체. 시퀀스 객체가 대표적이다.

논리 연산자

참과 거짓을 다루는 연산자

—and: 둘 다 참이면 True

```
>>> True and True
True
>>> True and False
False
```

```
>>> False and True
False
>>> False and False
False
```

—or: 둘 중 하나만 참이면 True

```
>>> True or True
True
>>> True or False
True
```

```
>>> False or True
True
>>> False or False
False
```

논리 연산자

참과 거짓을 다루는 연산자

—not: 참을 거짓으로 거짓을 참으로

```
>>> not True  
False  
>>> not False  
True
```

논리 연산자의 순서

여러 논리 연산자가 들어있을 때 순서

—not, and, or 순으로 판단

```
>>> not True and False or not False
True
>>> ((not True) and False) or (not False)
True
```

—순서가 헷갈리면 괄호로 판단 순서를 명확히 나타낸다.

비교연산자

주어진 두 값을 비교하여 참과 거짓을 반환

| 비교 연산자 | 설명 |
|-----------|----------------|
| $x==y$ | x와 y가 같다. |
| $x!=y$ | x와 y가 같지 않다. |
| $x>y$ | x가 y보다 크다. |
| $x\geq y$ | x가 y보다 크거나 같다. |
| $x<y$ | x가 y보다 작다. |
| $x\leq y$ | x가 y보다 작거나 같다. |

중첩 비교연산자

범위를 지정

- 변수가 0보다 크고, 10보다 작으면,
'10보다 작은 양수입니다.'를 출력

```
>>> x = 5
>>> if x > 0 and x < 10:
...     print('10보다 작은 양수입니다.')
...
10보다 작은 양수입니다.
```

```
10보다 작은 양수입니다.
>>> x = 3
>>> if 0 < x < 10:
...     print('10보다 작은 양수입니다.')
...
10보다 작은 양수입니다.
```

연산자 in & not in

특정 값이 있는지 확인하는 연산자(in)

— 값 in 시퀀스객체

```
>>> a
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> 40 in a
True
>>> 25 in a
False
>>> a = "Good Moring"
>>> "Goo" in a
True
>>> "ood" in a
True
>>> "Mro" in a
False
```

연산자 in & not in

특정 값이 없는지 확인하는 연산자(not in)

—값 not in 시퀀스객체

```
>>> a
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> 25 not in a
True
>>> 40 not in a
False
```

내장함수-len()

요소의 개수를 구한다. (len(객체))

—리스트

```
>>> a
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> len(a)
10
```

—튜플

```
>>> a = tuple(a)
>>> a
(0, 10, 20, 30, 40, 50, 60, 70, 80, 90)
>>> len(a)
10
```

—문자열

```
>>> a = "Python"
>>> len(a)
6
```

```
>>> a = "내장함수"
>>> len(a)
4
```

기본문법-주석(#)

주석(Comment, #)

- '어떤 낱말이나 문장을 쉽게 풀이한 글'
- 인터프리터가 처리하지 않아 프로그램의 실행에는 영향을 주지 않음

- 한 줄 주석

```
>>> #여기는 주석입니다.  
... print("abc")  
abc
```

- 블록 주석

```
>>> #Author: 김재형  
... #Date: 2018.12.11  
... print("abc")  
abc
```

기본문법-주석(#)

주석(Comment, #)

—인라인(inline) 주석

```
>>> print("abc") #여기는 주석입니다.  
abc
```

—쓰지 않는 것을 추천

※ 외국에서는 #을 해시(hash), 샤프(sharp), 파운드(pound), 옥토쏘르프(octothorpe) 등으로 부른다.

PEP8

-Python 코드 스타일 가이드

Comments(주석)

- 코드에 따라 주석은 갱신되어야 한다.
- 불필요한 주석은 달지 말기
- 한 줄 주석은 신중히
- Docsting

기본문법-주석(#)

주석달기

- 달지 말아야 되는 것
 - 한눈에 보기에 명확한 내용
 - 코드에서 빠르게 유추할 수 있는 내용(변수명 등)
- 달아야 되는 것
 - 자신의 생각을 기록
 - 어떤 생각으로? 어떤 점을 깨달았는지?
 - 자신이 읽는 사람이 되어서 경고
읽다가 바로 이해되지 않는 위치

기본문법-주석(#)

주석달기

- 명확하고 간결한 주석 달기
 - 최대한 간결하고 명확하게 뜻을 전달하라
 - 대명사를 쓰는 것은 좋지 않다.
- 함수의 동작을 설명
 - ex) 이 파일의 담긴 줄 수를 반환한다.
=> 파일 내부에서 새 줄을 나타내는 $\backslash n$ 의 개수를 센다.
- 코드의 의도를 명시
 - 개발자가 생각했던 의도를 작성하는 것이 중요하다.

기본문법-세미콜론(;)

세미콜론(;)

- 많은 프로그래밍 언어는 구문이 끝날 때 붙임
- Python는 사용하지 않는다.

```
>>> print("Hello, world")  
Hello, world
```

- 세미콜론을 통해 여러 구문을 한 줄에 사용가능
※ 추천하지는 않음

```
>>> print("hello, world"); print("Python is easy")  
hello, world  
Python is easy
```

기본문법-백슬래시(\)

백슬래시(\, 키보드: ₩)

- 라인을 유지할 때 사용
- PEP8에서는 코드를 읽기 편하게 하기 위해 79자를 추천
- 여러 줄에 걸쳐서 한 문장을 입력해야 될 때가 있음
- 단 괄호내에서 줄을 나눌 때는 백슬래시를 쓰지 않는다.

```
>>> alphabet = 'abcdefg'\  
...           + 'hijklmnop'\  
...           + 'qrstuvwxyz'  
>>> alphabet  
'abcdefghijklmnopqrstuvwxyz'
```

```
>>> 1+2+3+4+\  
... 5+6+7  
28
```

```
>>> print("hello"  
...      "hello"  
...      "hi!")  
hellohellohi!
```

조건문(분기문, if)

구조화 프로그래밍

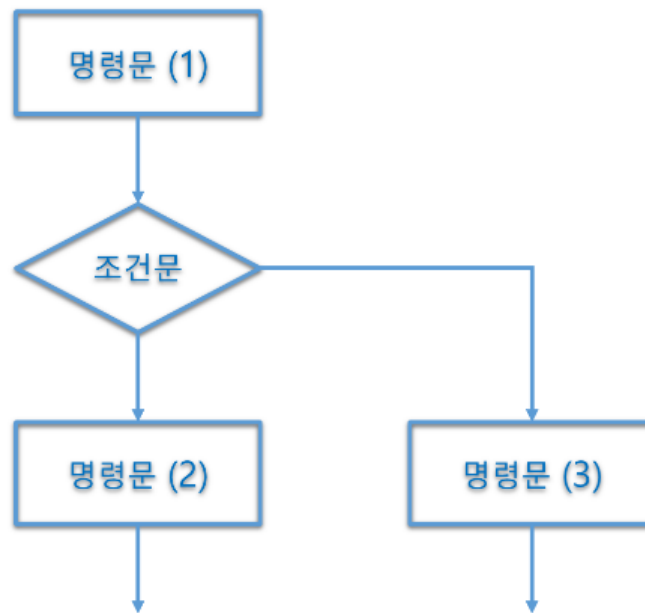
- 1960년대 후반
 - "사람이 프로그램을 보다 편하게 쓰고 읽을 수 있도록 규칙을 만들자!"
- if, while문 등이 이때 도입되었다

조건문(분기문, if)

특정 조건일 때 코드를 실행

—ex) if 12:00이 되면:
 점심을 먹는다.

이를 통해 프로그램의 흐름을 바꾼다.



조건문(분기문, if)

어셈블리어-if가 없었을 때?

- Jump 명령을 사용
 - 8(%rbp): 원래 코드의 x
 - %eax: 임시저장소
 - cmpl: 비교
 - Jne: 동일하지 않으면 jump

```
x = 123
# if문 앞
if x == 234:
    #if문 내부
#if문 뒤
```

```
_main:
.....
movl $123, -8(%rbp)
# if문 앞
movl -8(%rbp), %eax
cmpl $456, %eax
jne LBB1_2
# if문 안
LBB1_2:
# if문 뒤
```















조건문(분기문, if)

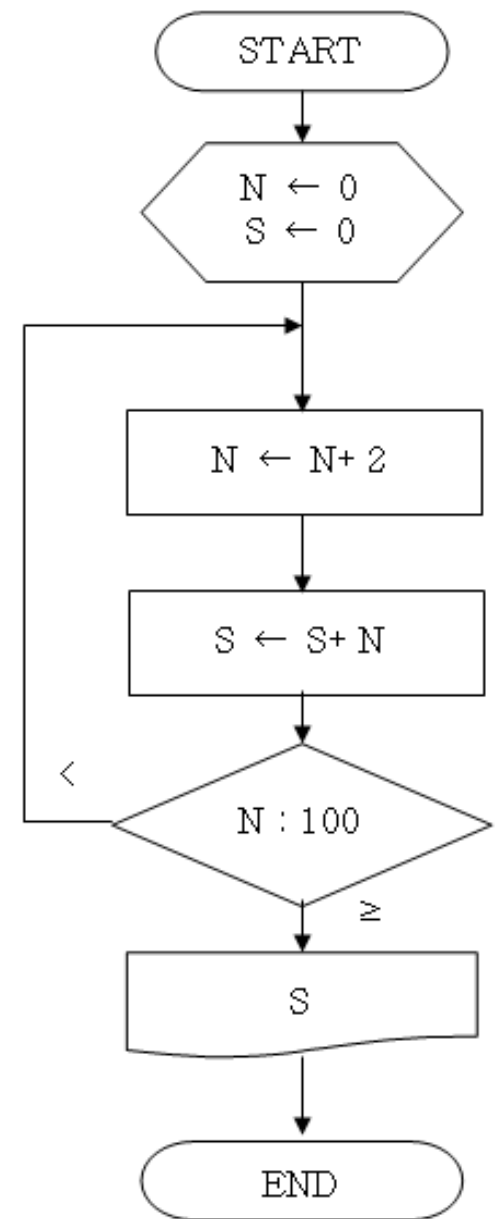
- 이렇게 조건을 만족하는 점프하는 명령은 1949년에도 있었다.
- 이를 보기 쉽게 만든 것이 if이다.

순서도

여러 종류의 상자과 이를 이어주는 화살표로 순서를 보여주는 것

■ 순서도 기호 ■

| | | | | | |
|--|-----|------------------------------------|---|-----------|---------------------------|
|  | 단말 | 순서도의 시작과 끝 |  | 카드입력 | 카드리더(card reader)를 통한 입력 |
|  | 흐름선 | 작업 흐름을 명시 |  | 수동입력 | 키보드를 통한 입력 |
|  | 준비 | 작업 단계 시작 전 준비 (변수 및 초기치 선언 등) |  | 서브루틴 | 정의하여 둔 부프로그램의 호출 |
|  | 처리 | 처리하여야 할 작업을 명시 (변수에 계산 값 입력 등) |  | 페이지 내 연결자 | 한 페이지 내의 순서도 연결 |
|  | 입출력 | 일반적인 데이터의 입력 또는 결과의 출력 |  | 페이지 간 연결자 | 페이지가 다른 순서도의 연결 |
|  | 판단 | 조건에 따라 흐름선을 선택 (일반적으로 참, 거짓 구분) |  | 화면표시 | 처리결과 또는 메시지를 모니터를 이용하여 출력 |
|  | 프린트 | 프린터를 이용한 출력 (서류 등의 지면에 출력) |  | 결합 | 기본 흐름선에 다른 흐름선 합류 |



조건문(분기문, if)

if문의 문법

if 조건문:

수행할 문장1

수행할 문장2

기본문법-들여쓰기

들여쓰기

- 코드를 읽기 쉽도록 일정한 간격을 띄워서 작성
- 파이썬은 들여쓰기 자체가 문법이다.

```
>>> if a == 10:
... print('10입니다.')
    File "<stdin>", line 2
      print('10입니다.')
      ^
```

```
IndentationError: expected an indented block
```

```
>>> if a == 10:
...     print('10입니다.')
... 
```

- Tab이나 공백4칸을 사용할 수 있으나 혼합은 불가능하다
- PEP8에 따라 공백4칸을 사용하기로 한다.

기본문법-코드 블록

코드블록

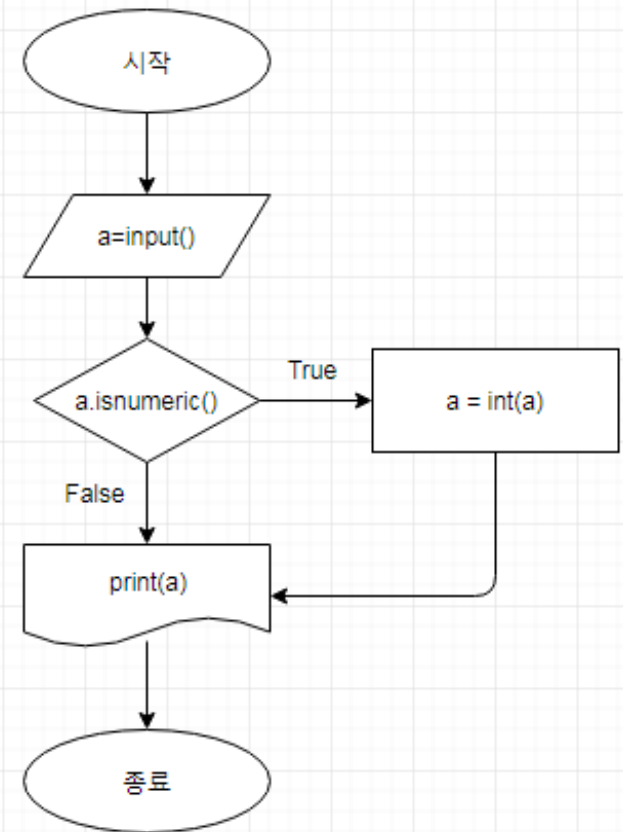
- 여러 코드가 이루는 일정한 구역
- 특정한 동작을 위해서 코드가 모여있는 상태
- 들여쓰기를 기준으로 코드블록을 구성
- 단, 공백과 탭 문자를 섞으면 안된다.

```
>>> a=3
>>> if a == 3:
...     print('삼입니다.')
...     print('Three!')
... else:
...     print('삼이 아닙니다.')
...
삼입니다.
Three!
```

조건문(분기문, if)

if문의 문법

- 입력한 값이 숫자면 정수형으로 바꾸는 프로그램



조건문(분기문, if)

if문의 문법

- 입력한 값이 숫자면 정수형으로 바꾸는 프로그램

```
1  input_value = input("값을 입력하세요: ")
2  if input_value.isnumeric():
3      input_value = int(input_value)
4  print(input_value)
5
```

조건문(분기문, if)

if문의 중첩

- 프로그래밍을 하다 보면 여러 조건을 고려해야 되는 경우가 있다.

if 조건문:

수행문

if 조건문:

수행문

조건문(분기문, if)

if문의 중첩

- 받은 값이 10이상이면 '10 이상입니다.' 출력
- 이 때 값이 20이면 '20입니다.' 출력

```
1 value = input("값을 입력해주세요: ")
2 if value >= 10:
3     print("10 이상입니다.")
4     if value == 20:
5         print("20입니다.")
```

조건문(분기문, if)

else문

- if 블록 밖의 문장은 항상 실행된다.
- 조건문이 거짓이면 다른 문장을 실행하고 싶다.

if 조건문:

수행할 문장1

수행할 문장2

...

else:

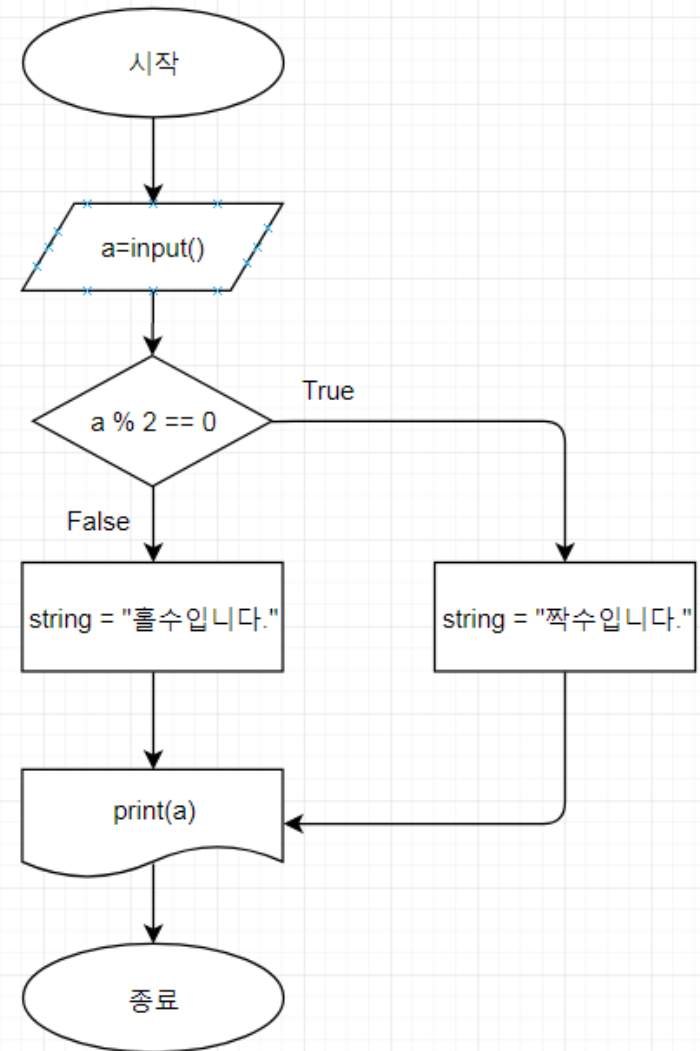
수행할 문장1

...

조건문(분기문, if)

else문

- 홀짝 판별 프로그램
- 입력받은 값이 홀수이면 "홀수입니다."
짝수이면 "짝수입니다." 를 출력한다.



조건문(분기문, if)

else문

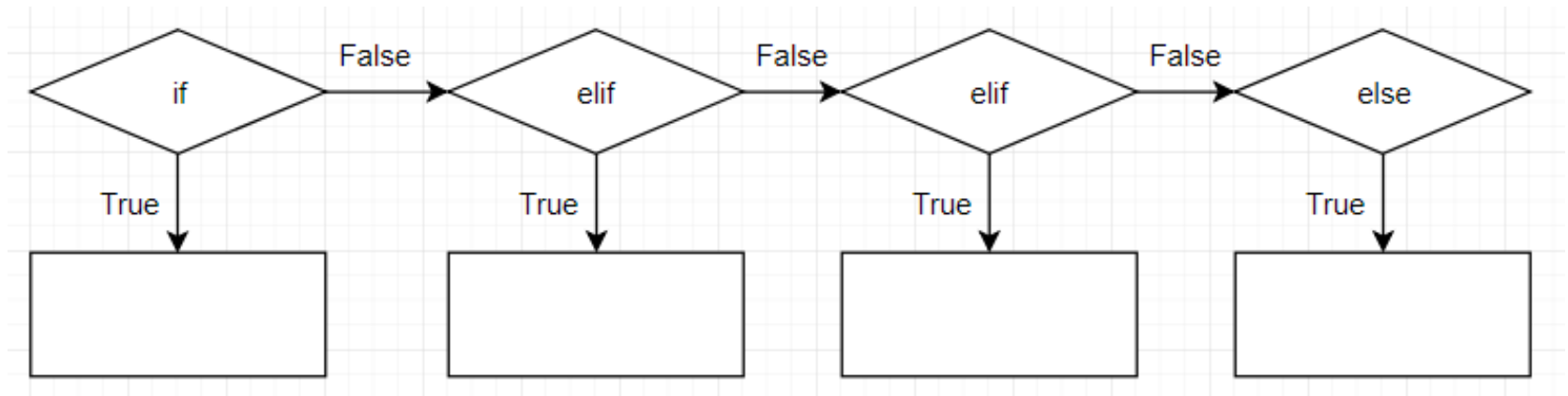
—홀짝 판별 프로그램

```
1 value = input("값을 입력해주세요: ")
2 if value % 2 == 0:
3     string = "짝수입니다."
4 else:
5     string = "홀수입니다."
6 print(string)
7
```

조건문(분기문, if)

elif문

- 여러 조건을 확인하고 싶다!
- elif를 if와 else 사이에 넣어 다른 조건을 부여



조건문(분기문, if)

elif문

if 조건문:

수행문

elif 조건문2:

수행문

....

else:

수행문

조건문(분기문, if)

학점 ABC매기기

- 80점 이상 A
- 60점 이상 B
- 그 외에 C
- 단, 입력값은 0-100사이이다.

```
1 value = int(input("학점을 입력하세요: "))
2
3 if value > 80:
4     print("A")
5 elif value > 60:
6     print("B")
7 else:
8     print("C")
```

조건문(분기문, if)

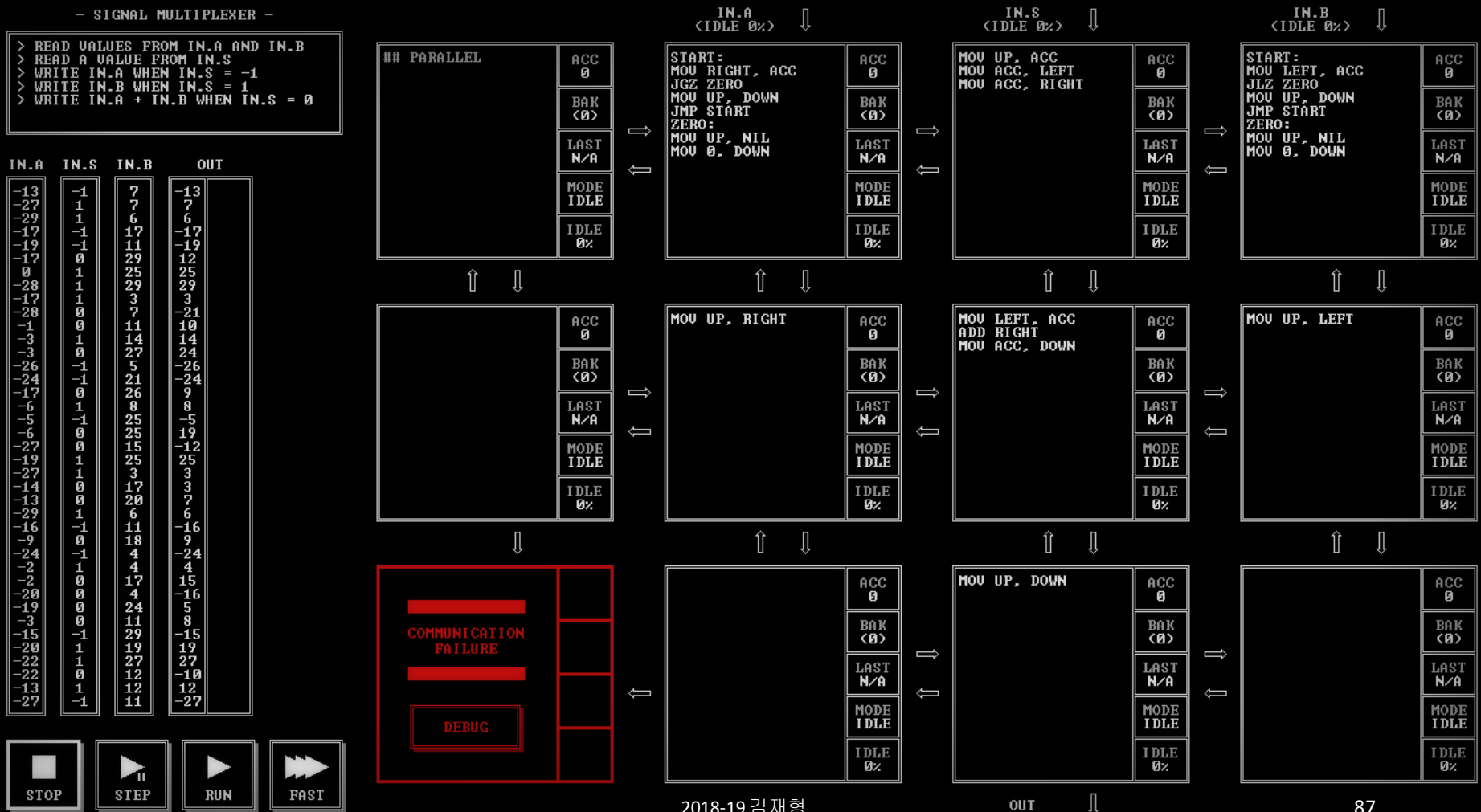
else, elif가 없었을 때?

- jl(jump if less '<')
- jg(jump if greater '>')
- Jmp(jump)

```
# if문 앞
if x > 0:
    # 양수일 경우 처리
elif x < 0:
    # 음수일 경우 처리
else:
    # 0일 경우 처리
# if문 뒤
```

```
_main:
    .....
    # if문 앞
    movl    -8(%rbp), %eax
    cmpl    $0, %eax
    jl      LBB1_2
    # 양수일 경우 처리
    jmp     LBB1_5
LBB1_2:
    movl    -8(%rbp), %eax
    cmpl    $0, %eax
    jg      LBB1_4
    # 음수일 경우 처리
    jmp     LBB1_5
LBB1_4:
    # 0일 경우 처리
LBB1_5:
    # if문 뒤
```

조건문(분기문, if)



조건문(분기문, if)

c언어의 goto문/if ~ elif ~ else의 좋은 점

- goto문은 코드를 복잡하게 만들기 때문에 특수한 경우를 제외하고는 안 쓰는 것이 좋다.
- Python에는 goto문에 없다.

```
void not_use_if(int x){
    if(x <= 0) goto NOT_POSITIVE;
    printf("양수\n");
    goto EN;
NOT_POSITIVE:
    if(x >= 0) goto NOT_NEGATIVE;
    printf("음수\n");
    goto END;
NOT_NEGATIVE:
    print("0\n");
END:
    return;
}
```

```
void use_if(int x){
    if(x > 0){
        printf("양수\n");
    }
    else if(x < 0){
        printf("음수\n");
    }
    else{
        print("0\n");
    }
}
```


디버깅

버그

- 소프트웨어가 예상한 동작을 하지 않고 잘못된 결과를 내거나, 오류가 발생하거나 작동이 실패하는 등의 문제를 뜻한다.



디버깅

버그

- 오타
- 특수한 케이스 미 고려
- OS에 의한 오류 등

- 슈뢰딩거버그: 다른걸 고치니 오류가 사라짐
- 하이젠버그: 디버깅툴을 쓰니 버그가 사라짐
- 나비효과: A->B->C->A...



수정하지 않았는데 사라지는 버그 끝판왕이 강림했습니다
스마트폰 게임개발이야기 35화

디버깅

디버깅: 버그를 찾고 수정

- 20%가 코딩이면 80%가 디버깅이라는 이야기가 있다.
- 오류 혹은 비정상적인 작동을 하는 부분을 찾아 수정
- 변수의 각 단계별 변화를 확인하면서 테스트

디버깅

디버깅툴(디버거)가 없을 때.

- 확인할 변수를 print하면서 변수의 변화를 확인.
- input()과 같이 중간에 멈출 수 있는 함수를 넣음

디버깅

디버깅툴(디버거)가 있을 때,

- 확인할 부분의 중단점을 잡아준다.
- 이후 한 step씩 진행하면서 변수의 값의 변화를 본다.

기본과제-자판기2

vending_machine.py

- 고객이 자판기에 입력한 돈으로 물품을 주길 원하는다.
- 이전의 프로그램에 추가한다.

기본과제-자판기2

vending_machine.py

- 1. 넣은 돈을 출력한 뒤,
- 2. "뽑을 물품을 골라주세요: "를 출력하고 뽑을 물품번호를 입력받는다.
- 3. 고객이 물품 번호를 잘못 입력할 경우, "물품번호를 잘못 입력하셨습니다."를 출력하고 돈을 반환한 뒤 종료한다.
- 4. 거스름돈을 선택하면, 돈을 반환한다

기본과제-자판기2

vending_machine.py

- 5. 물품 번호를 정확히 입력했을 경우
 - 5.1 물품 값보다 넣은 돈이 많으면, "(선택한 물품) 이/가 나왔습니다."를 출력한 후, 물품 값을 뺀 나머지 값을 돌려준다.
 - 5.2 물품 값보다 넣은 돈이 적으면 "돈이 부족합니다"를 출력하고 돈을 반환한다.
- 단, 입력받는 값은 정수이다.

기본과제-자판기2

vending_machine.py

—예시: 물품 번호를 잘못 입력하였을 때

```
돈을 넣으세요 : 100
1. 블랙커피 (100원)
2. 밀크커피 (150원)
3. 고급커피 (200원)
4. 거스름돈
넣은 돈 : 100원
뽑을 물품을 골라주세요 : 5
물품번호를 잘못 입력하셨습니다.
돈을 반환합니다.: 100원
```

기본과제-자판기2

vending_machine.py

—예시: 거스름돈을 선택하였을 때

```
돈을 넣으세요 : 2000
1. 블랙커피 (100원)
2. 밀크커피 (150원)
3. 고급커피 (200원)
4. 거스름돈
넣은 돈 : 2000원
뽑을 물품을 골라주세요 : 4
돈을 반환합니다. : 2000원
```

기본과제-자판기2

vending_machine.py

—예시: 돈이 부족할 때

```
돈을 넣으세요 : 50
1. 블랙커피 (100원)
2. 밀크커피 (150원)
3. 고급커피 (200원)
4. 거스름돈
넣은 돈 : 50원
뽑을 물품을 골라주세요 : 3
돈이 부족합니다.
돈을 반환합니다. : 50원
```

기본과제-자판기2

vending_machine.py

—예시: 돈이 충분할 때

```
돈을 넣으세요 : 150
1. 블랙커피 (100원)
2. 밀크커피 (150원)
3. 고급커피 (200원)
4. 거스름돈
넣은 돈 : 150원
뽑을 물품을 골라주세요 : 1
블랙커피이/가 나왔습니다.
돈을 반환합니다. : 50원
```

기본과제-BMI 계산기

BMI_calculator.py

- BMI, 체질량지수는 인간의 비만도를 나타내는 지수로 키와 몸무게로 간단히 추정할 수 있다. 단, 의사들도 근육량과 지방의 밀도등을 고려하지 않아 의문을 제기하는 지수이다.



기본과제-BMI 계산기

BMI_calculator

- 대한비만학회에 따르면,
18.5미만이면 '저체중',
18.5-23은 '정상',
23-25는 '과체중',
25-30은 '경도비만',
30-35는 '중증도 비만',
35이상이면 '고도 비만'으로 구분한다.

기본과제-BMI 계산기

BMI_calculator.py

- BMI계산은 다음과 같다.
 - 몸무게/키²(몸무게를 키의 제곱으로 나눈 것)
 - 단, 몸무게는 kg, 키는 m단위이다.
 - BMI를 계산해서 판단해주는 프로그램을 만드시오.
 - 자세한 출력 및 입력 사항은 본인의 판단에 따라 생성하고, 원하는 자료형으로 입력받는 것으로 생각한다.
- ※ `abs(x)`는 숫자형을 받아 절대값을 되돌려준다.

기본과제-BMI 계산기

BMI_calculator.py

—예시

```
본인의 키를 입력하세요.(m): 1.66
본인의 몸무게를 입력하세요.(kg): 74
경도비만
root@goorm:/workspace/PythonSeminar18/Ex:
n3 BMI_calculator.py
본인의 키를 입력하세요.(m): 1.8
본인의 몸무게를 입력하세요.(kg): 74
정상
root@goorm:/workspace/PythonSeminar18/Ex:
n3 BMI_calculator.py
본인의 키를 입력하세요.(m): 1.66
본인의 몸무게를 입력하세요.(kg): 85
중증도 비만
```