프로그래밍

이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

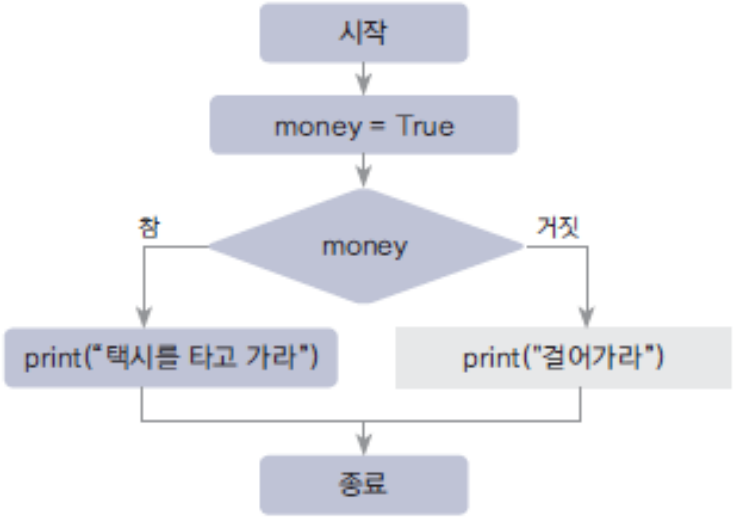
제어문
(if문, while문, for문)

if문

- 프로그래밍에서 조건을 판단하여 해당 조건에 맞는 상황을 수행할 때 'if문'이 사용됨

'돈이 있으면 택시를 타고 가고, 돈이 없으면 걸어간다.'

```
>>> money = True
>>> if money:
...     print("택시를 타고 가라")
... else:
...     print("걸어 가라")
...
택시를 타고 가라
```



if문

- if문의 기본 구조
  - 조건문을 테스트해서 참이면 if문 바로 다음 블록을 수행하고, 조건문이 거짓이면 else 문 다음 블록을 수행함

```
if 조건문:
    수행할_문장1
    수행할_문장2
    ...
else:
    수행할_문장A
    수행할_문장B
    ...
```

if문

- if문의 기본 구조
  - if문을 만들 때는 'if 조건문:(혹은 else:)' 바로 다음 문장부터 if문에 속하는 모든 문장에 들여쓰기(indentation)를 해야함

```
if 조건문:
    수행할_문장1
    수행할_문장2
    수행할_문장3
```

```
if 조건문:
    수행할_문장1
수행할_문장2
    수행할_문장3
```

〈오류발생〉

```
if 조건문:
    수행할_문장1
    수행할_문장2
        수행할_문장3
```

if문

- 조건문
  - if 조건문에서 '조건문'이란 참과 거짓을 판단하는 문장을 말함

```
>>> money = True
>>> if money:
```

  - 예제에서 조건문은 money이며, money는 True이기 때문에 참이 되어 if문 다음 문장을 수행

  - 조건문에는 비교 연산자(<, >, ==, !=, >=, <=), 불 연산자(and, or, not) 그리고 in 연산자가 주로 사용됨

if문

- 비교 연산자

만약 3000원 이상의 돈을 가지고 있으면 택시를 타고 가고, 그렇지 않으면 걸어가라.

```
>>> money = 2000
>>> if money >= 3000:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
걸어가라
>>>
```

if문

- 불 연산자

연산자	설명
x or y	x와 y 둘 중 하나만 참이어도 참이다.
x and y	x와 y 모두 참이어야 참이다.
not x	x가 거짓이면 참이다.

if문

- 불 연산자

돈이 3000원 이상 있거나 카드가 있다면 택시를 타고 가고, 그렇지 않으면 걸어가라.

```
>>> money = 2000
>>> card = True
>>> if money >= 3000 or card:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```

if문

- in 연산자

```
>>> 1 in [1, 2, 3]
True
>>> 1 not in [1, 2, 3]
False
```

```
>>> 'a' in ('a', 'b', 'c')
True
>>> 'j' not in 'python'
True
```

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

if문

- in 연산자

```
만약 주머니에 돈이 있으면 택시를 타고 가고, 없으면 걸어가라.
```

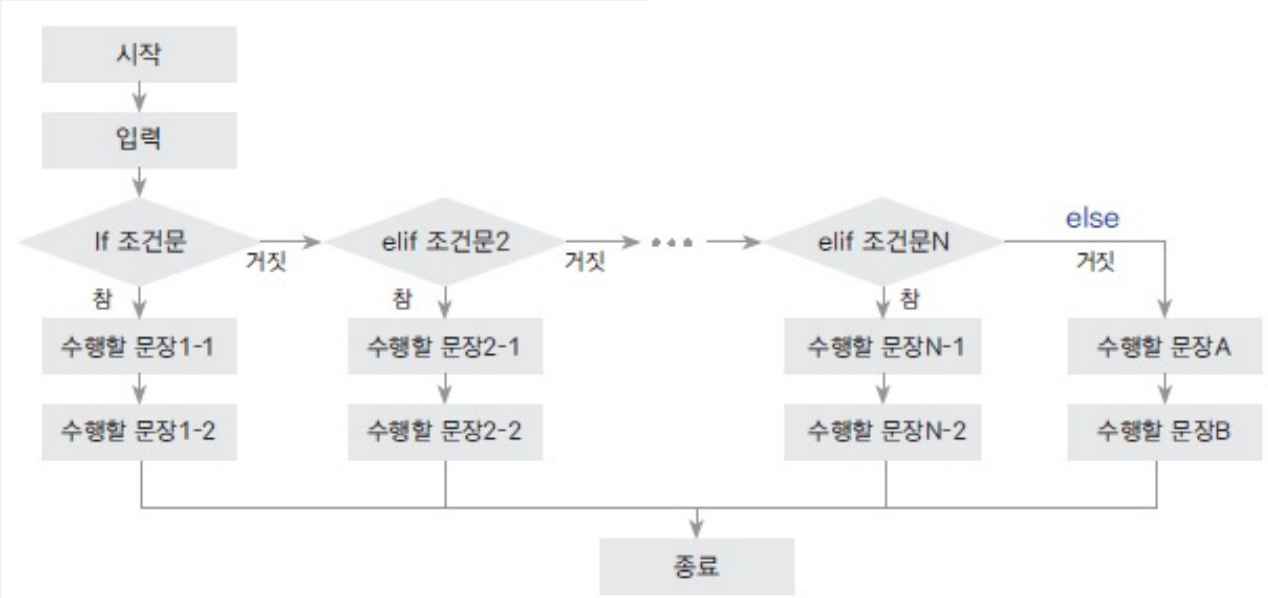
단, 주머니 pocket=['paper', 'cellphone', 'money']로 구성

```
>>> pocket = ['paper', 'cellphone', 'money']
>>> if 'money' in pocket:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```

if문

- elif문
  - 다양한 조건을 판단하기 위해서 elif문을 사용함
  - elif는 개수에 제한 없이 사용할 수 있음

```
if 조건문:
    수행할_문장1
    수행할_문장2
    ...
elif 조건문:
    수행할_문장1
    수행할_문장2
    ...
elif 조건문:
    수행할_문장1
    수행할_문장2
    ...
...
else:
   수행할_문장1
   수행할_문장2
   ...
```



if문

- elif문

주머니에 돈이 있으면 택시를 타고 가고, 주머니에 돈은 없지만 카드가 있으면 택시를 타고 가고, 돈도 없고 카드도 없으면 걸어가라.

```
>>> pocket = ['paper', 'cellphone']
>>> card = True
>>> if 'money' in pocket:
...       print("택시를 타고가라")
... elif card:
...       print("택시를 타고가라")
... else:
...       print("걸어가라")
...
택시를 타고가라
```

if문

- **조건부 표현식**

```
변수 = 조건문이_참인_경우의_값 if 조건문 else 조건문이_거짓인_경우의_값
```

```
if score >= 60:
    message = "success"
else:
    message = "failure"
```

```
message = "success" if score >= 60 else "failure"
```

while문

- while문 기본 구조
  - while문은 조건문이 참인 동안 while문에 속한 문장들이 반복해서 수행

```
while 조건문:
    수행할_문장1
    수행할_문장2
    수행할_문장3
    ...
```

while문

- while문 기본 구조

```
>>> treeHit = 0
>>> while treeHit < 10:
...     treeHit = treeHit +1
...     print("나무를 %d번 찍었습니다." % treeHit)
...     if treeHit == 10:
...         print("나무 넘어갑니다.")
...
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

treeHit	조건문	조건 판단	수행하는 문장	while문
0	0 < 10	참	나무를 1번 찍었습니다.	반복
1	1 < 10	참	나무를 2번 찍었습니다.	반복
2	2 < 10	참	나무를 3번 찍었습니다.	반복
3	3 < 10	참	나무를 4번 찍었습니다.	반복
4	4 < 10	참	나무를 5번 찍었습니다.	반복
5	5 < 10	참	나무를 6번 찍었습니다.	반복
6	6 < 10	참	나무를 7번 찍었습니다.	반복
7	7 < 10	참	나무를 8번 찍었습니다.	반복
8	8 < 10	참	나무를 9번 찍었습니다.	반복
9	9 < 10	참	나무를 10번 찍었습니다. 나무 넘어갑니다.	반복
10	10 < 10	거짓		종료

while문

- 무한 루프
  - 무한 루프란 무한히 반복한다는 의미로, while문의 조건문이 항상 True일 경우 발생함

```
while True:
    수행할_문장1
    수행할_문장2
    ...
```

while문

- 무한 루프
  - 많은 일반 프로그램 중에서 무한 루프 개념을 사용

```
print('1. choice 1')
print('2. choice 2')
print('3. exit')

while True:
        ch = int(input())
        if ch == 1:
                print('choice 1')
        elif ch == 2:
                print('choice 2')
        elif ch == 3:
                print('exit')
                break
```

  - 조건문을 잘못 설정하여 아무 의미 없이 무한 루프를 돌리는 것을 주의!

while문

- break문
  - while문은 조건문이 참인 동안 계속 while문 안의 내용을 반복적으로 수행한다. 하지만 강제로 while문을 빠져나가고 싶을 때 'break문'을 사용

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee -1
...     print("남은 커피의 양은 %d개입니다." % coffee)
...     if coffee == 0:
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
...
```

while문

- continue문
  - while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우, 'continue문'을 사용

  - 1부터 10까지의 숫자 중에서 홀수만 출력하는 것을 while문으로 작성한다면?

```
>>> a = 0
>>> while a < 10:
...     a = a + 1
...     if a % 2 == 0: continue
...     print(a)
...
1
3
5
7
9
```

for문

- for문의 기본 구조
  - 리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 '수행할_문장1', '수행할_문장2' 등이 수행

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할_문장1
    수행할_문장2
    ...
```

for문

- for문의 기본 구조

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:
...     print(i)
...
one
two
three
```

```
>>> a = [(1,2), (3,4), (5,6)]
>>> for (first, last) in a:
...     print(first + last)
...
3
7
11
```

for문

- for문의 기본 구조

총 5명의 학생이 시험을 보았는데 시험 점수가 60점 이상이면 합격이고 그렇지 않으면 불합격이다. 합격인지, 불합격인지 결과를 보여 주시오.

- 5명의 시험 점수를 리스트로 표현하면 다음과 같음

```
marks = [90, 25, 67, 45, 80]
```

```
number = 0   # 학생에게 붙여 줄 번호
for mark in marks:   # 90, 25, 67, 45, 80을 순서대로 mark에 대입
    number = number +1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)
```

for문

- continue문
  - for문의 처음으로 돌아가길 원하면 continue문을 사용

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다. " % number)
```

  - continue문을 이용하여 60점 이상인 사람에게만 축하 메시지를 보내도록 변경

for문

- break문
  - for문을 중간에 종료하길 원하면 break문을 사용

```
marks = [90, 25, 67, 45, 80]

number = 1
for mark in marks:
        if mark < 60:
                continue
        print(f"{number}번 학생 축하합니다. 합격입니다.")
        if number == 2:
                break
        number += 1
```

for문

- range 함수
  - for문은 숫자 리스트를 자동으로 만들어 주는 range 함수와 함께 주로 사용됨

```
>>> a = range(10)
>>> a
range(0, 10)
```

  - range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 줌
  - 시작 숫자와 끝 숫자를 지정하려면 range(시작_숫자, 끝_숫자) 형태를 사용하는데, 이때 끝 숫자는 포함되지 않음

for문

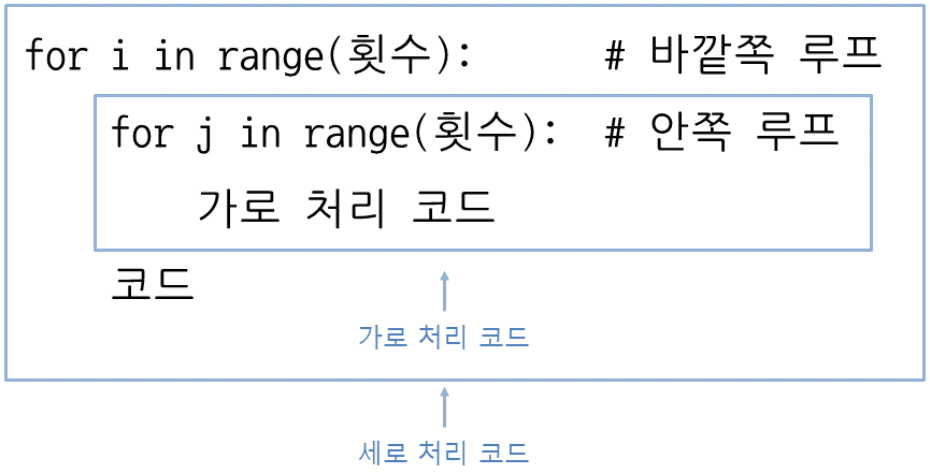
- range 함수

```
>>> add = 0
>>> for i in range(1, 11):
...     add = add + i
...
>>> print(add)
55
```

```
for number in range(len(marks)):
    if marks[number] < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다." % (number+1))
```

for문

- 중첩 루프(nested loops)
  - 반복 구조의 본체가 또 다른 반복 구조를 포함한 경우에 이러한 반복 구조를 중첩 루프(nested loops)라 부름



for문

- 중첩 루프(nested loops)
  - for문과 range 함수를 사용하여 구구단을 출력하는 프로그램을 작성하시오.

```
>>> for i in range(2,10):          # 1번 for문
...     for j in range(1, 10):     # 2번 for문
...         print(i*j, end=" ")
...     print('')
...
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

for문

- 리스트 컴프리헨션
  - 리스트 안에 for문을 사용하여 리스트 내 요소값들에 반복적인 연산이 가능하게 함

```
>>> a = [1,2,3,4]
>>> result = []
>>> for num in a:
...     result.append(num*3)
...
>>> print(result)
[3, 6, 9, 12]
```

```
>>> a = [1,2,3,4]
>>> result = [num * 3 for num in a]
>>> print(result)
[3, 6, 9, 12]
```

Q&A