



프로그램의 흐름 제어하기

파이썬 프로그래밍

Contents

❖ 흐름 제어를 시작하기 전에

- bool 자료형
- 논리 연산자
- 흐름 제어문과 조건문
- 코드블록과 들여쓰기
- 비교 연산자

❖ 분기문

- if문

❖ 반복문

- while문
- for문
- continue와 break로 반복문 제어하기



흐름 제어를 시작하기 전에 - bool 자료형

❖ bool 자료형

- bool은 True와 False 두 가지 값을 나타내는 자료
- 실습 1

```
>>> a = 3 > 2
>>> a
True
>>> a = 2 > 3
>>> a
False
>>> type(a)
<class 'bool'>
```



흐름 제어를 시작하기 전에 - 논리 연산자

❖ 논리 연산자

• not 연산자 - 1

- 피연산자를 부정. 피연산자가 False인 경우에는 True, 그렇지 않은 경우에는 False 출력
- 실습 1

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

- bool 형식 외에도, 수, 문자열, 튜플, 리스트, 딕셔너리도 피연산자로 사용 가능
- 입력된 피연산자가 0인 경우에는 피연산자를 거짓으로 간주. not 연산의 결과는 True
- 0이 아닌 수를 피연산자로 사용하는 경우의 not 연산 결과는 False
- 실습 2

```
>>> not 0
True
>>> not -1
False
>>> not 1
False
```

흐름 제어를 시작하기 전에 - 논리 연산자

❖ 논리 연산자

• not 연산자 - 2

- None에 대해서도 거짓으로 간주하기 때문에 not None의 결과는 True
- 실습 3

```
>>> not None
True
```

- not 연산자는 비어있는 문자열이나 튜플, 리스트, 딕셔너리도 거짓으로 간주

```
>>> not 'ABC' # 비어있지 않은 문자열을 부정
False
>>> not '' # 빈 문자열을 부정
True
>>> not (1, 2, 3) # 비어있지 않은 튜플을 부정
False
>>> not () # 빈 튜플을 부정
True
>>> not [] # 빈 리스트를 부정
True
>>> not {} # 빈 딕셔너리를 부정
True
```

흐름 제어를 시작하기 전에 - 논리 연산자

❖ 논리 연산자

• and 연산자

- 두 피연산자 간의 논리곱을 수행
- 논리곱 연산의 결과는 두 피연산자 모두가 True인 경우에만 True가 되고, 그렇지 않은 경우에는 항상 False
- 실습 1

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

• or 연산자

- 두 피연산자 모두가 False인 경우에만 False가 되고, 그렇지 않은 경우에는 그 결과가 항상 True
- 실습 1

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

흐름 제어를 시작하기 전에 - 흐름 제어문과 조건문

❖ 흐름 제어문과 조건문

- 흐름 제어문은 흐름을 분기하거나 반복하기 전에 조건문의 결과가 참인지를 평가 수행
- 조건문이 구체적으로 다음과 같을 때 거짓으로 평가
 - False
 - None
 - 숫자 0 예) 0, 0.0 등
 - 비어있는 순서열 : 예) "", (), [] 등
 - 비어있는 딕셔너리 : 예) {}
- 어떤 객체가 거짓으로 평가되는지를 알고 싶을 때는 bool() 함수 이용

```
>>> bool(False)
False
>>> bool(None)
False
>>> bool(0)
False
>>> bool(0.0)
False
>>> bool('')
False
>>> bool('Hello')
True
```

```
>>> bool(123)
True
>>> bool(())
False
>>> bool([])
False
>>> bool([1, 2, 3])
True
>>> bool({})
False
```



흐름 제어를 시작하기 전에 - 코드블록과 들여쓰기

❖ 코드블록(Code Block)

- 여러 코드가 이루는 일정한 구역
- 프로그래밍 언어들마다 이것을 표현하는 방법이 다름
- 파이썬은 들여쓰기로 구역을 나눔

```
if a == 3: # a가 3과 같다면
```

```
-----> print('삼')
```

```
-----> print('三')
```

```
-----> print('Three')
```

```
else:
```

```
-----> print('삼이 아님.')
```

코드-블록

코드 블록

- 들여쓰기는 스페이스(Space)나 탭(Tab) 둘 다 사용 가능
 - 파이썬을 만든 귀도는 PEP(Python Enhancement Proposals)-8에서 **스페이스 4칸**을 사용할 것을 권장



흐름 제어를 시작하기 전에 - 비교 연산자

❖ 비교 연산자

- 두 피연산자를 비교하는 연산자
- 파이썬은 다음 표와 같이 6가지를 제공

연산자	설명
<code>==</code>	양쪽에 위치한 피연산자가 서로 같으면 True, 그렇지 않으면 False입니다. >>> a = 30 >>> a == 30 True >>> a == 40 False
<code>!=</code>	양쪽에 위치한 피연산자가 서로 다르면 True, 그렇지 않으면 False입니다. >>> a = '안녕' >>> a != '안녕' False >>> a != 'Hello' True
<code>></code>	왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 크면 True, 그렇지 않으면 False입니다. >>> a = 30 >>> a > 20 True >>> a > 40 False



흐름 제어를 시작하기 전에 - 비교 연산자

연산자	설명
<code>>=</code>	<p>왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 크거나 같으면 True, 그렇지 않으면 False입니다.</p> <pre>>>> a = 30 >>> a >= 20 True >>> a >= 30 True >>> a >= 40 False</pre>
<code><</code>	<p>왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 작으면 True, 그렇지 않으면 False입니다.</p> <pre>>>> a = 30 >>> a < 40 True >>> a < 20 False</pre>
<code><=</code>	<p>왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 작거나 같으면 True, 그렇지 않으면 False입니다.</p> <pre>>>> a = 30 >>> a <= 40 True >>> a <= 30 True >>> a <= 20 False</pre>



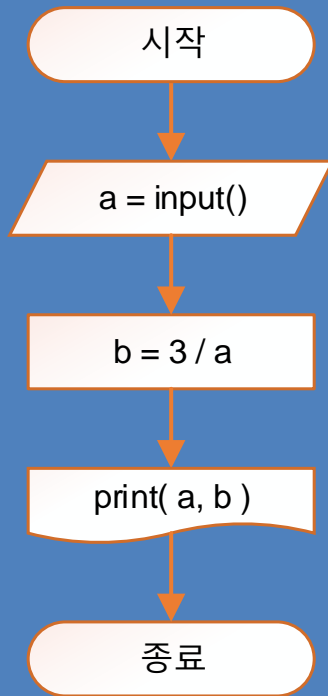
❖ 소프트웨어 ≡ 컴퓨터가 할 일의 목록

- 프로그래머가 컴퓨터에게 할 일의 목록을 내주면, 컴퓨터는 소프트웨어에 기록되어 있는 목록을 보고 그대로 수행
- 프로그래밍에서 “흐름(Flow)”은 컴퓨터에서 내려지는 명령의 순서를 가리키는 말
- 프로그램의 흐름을 가르는 문장 : 나눌 분(分), 갈림길 기(岐), 분기문

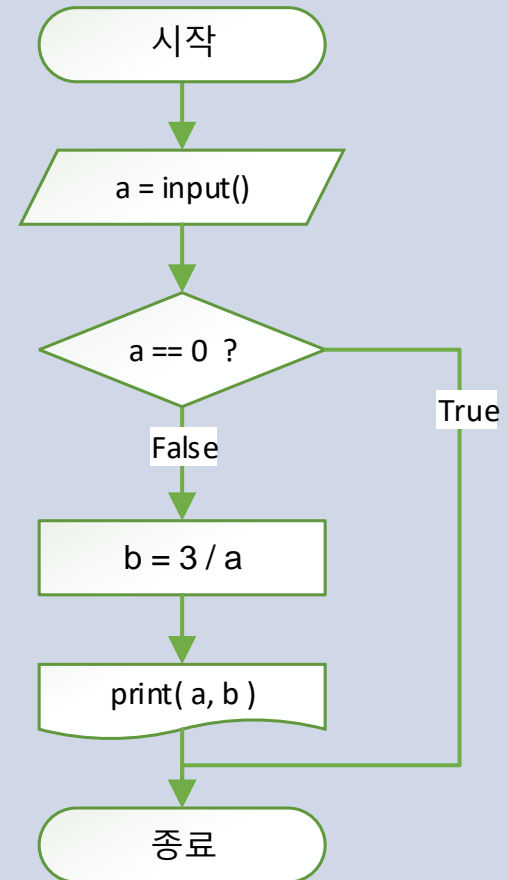
한 방향으로만 흐르는 프로그램	조건에 의해 흐름이 나뉘는 프로그램
<ol style="list-style-type: none">1. a에 사용자로부터 입력받은 수를 저장하라.2. b에 $3 \div a$의 결과를 저장하라.3. a와 b를 출력하라.	<ol style="list-style-type: none">1. a에 사용자로부터 입력받은 수를 저장하라.2. a가 0이라면 프로그램을 종료하고, 그렇지 않으면 다음 명령을 수행하라.3. b에 $3 \div a$의 결과를 저장하라.4. a와 b를 출력하라.



한 방향으로만 흐르는 코드



조건에 의해 흐름이 나뉘는 코드



분기문 - if문

- ❖ 영어에서 if문 “만약 ~ 라면 ” 의 뜻
 - 예) “만약 입력받은 수 a가 0이라면”
- ❖ if문의 조건은 참 아니면 거짓으로 평가될 수 있어야 함.
- ❖ 조건 뒤에 있는 콜론 : 은 해당 조건이 참일 경우에 실행할 코드 블록을 위치시키기 위함임.
- ❖ else 절로 코드가 흐르는 경우는 if절의 조건이 거짓으로 평가되었을 때

if 조건:

명령1

명령2

...

else:

명령3

명령4

...

if 뒤에 흐름을 가를 조건이 위치하고, 그 뒤에 콜론 : 이 옵니다.

: 뒤에는 들여쓰기로 이루어진 코드블록이 옵니다. 이 코드블록은 if문의 조건이 True일 경우 실행됩니다.

if문의 조건을 충족하지 않을 때, 즉 조건 평가의 결과가 False일 때의 흐름은 else로 향합니다. else 뒤에도 코드블록이 오므로 콜론 : 이 위치해야 합니다.



분기문 - if문

❖ 예제 : 06/ifelse.py

```
print('수를 입력하세요 : ')\na = int(input())\n\nif a == 0 :\n    print('0은 나눗셈에 이용할 수 없습니다.')\nelse :\n    print('3 /', a, '=', 3/a)
```

• 실행 결과

```
>ifelse.py\n수를 입력하세요 : \n0\n0은 나눗셈에 이용할 수 없습니다.
```

```
>ifelse.py\n수를 입력하세요 : \n12\n3 / 12 = 0.25
```



분기문 - if문

❖ 예제 : 06/if.py

```
import sys

print('수를 입력하세요 : ')
a = int(input())

if a == 0: # if not a: 와 동일한 코드
    print('0은 나눗셈에 이용할 수 없습니다.')
    sys.exit(0)

print('3 /', a, '=', 3/a)
```

• 실행 결과

```
>if.py
수를 입력하세요 :
0
0은 나눗셈에 이용할 수 없습니다.
```

```
>if.py
수를 입력하세요 :
8
3 / 8 = 0.375
```

분기문 - if문

❖ 여러 개의 조건을 다룰 때는?

- if와 함께 elif 절 사용

```
if 조건1:  
    코드블록  
    ...
```

첫 번째 조건은 항상 if로 시작합니다.

```
elif 조건2:  
    코드블록  
    ...
```

두 번째 조건부터는 elif를 이용합니다.

```
elif 조건3:  
    코드블록  
    ...
```

```
elif 조건4:  
    코드블록  
    ...
```

```
else:  
    코드블록  
    ...
```

마지막의 else는 생략할 수 있습니다.

분기문 - if문

❖ 예제 : 06/ifelif.py

```
print('요일(월~일)을 입력하세요 : ')
dow = input()

if dow == '월':
    print('Monday')
elif dow == '화':
    print('Tuesday')
elif dow == '수':
    print('Wednesday')
elif dow == '목':
    print('Thursday')
elif dow == '금':
    print('Friday')
elif dow == '토':
    print('Saturday')
elif dow == '일':
    print('Sunday')
else:
    print('잘못 입력된 요일입니다.')
```

• 실행 결과

```
>ifelif.py
요일(월~일)을 입력하세요 :
월
Monday
```

```
>ifelif.py
요일(월~일)을 입력하세요 :
수
Wednesday
```

```
>ifelif.py
요일(월~일)을 입력하세요 :
금
Friday
```

```
>ifelif.py
요일(월~일)을 입력하세요 :
사
잘못 입력된 요일입니다.
```



분기문 - if문

❖ 예제 : 06/ifif.py

```
print('수를 입력하세요 : ')\n a = int(input())
```

if a > 10 : 의 코드블록

```
if a > 10:
```

```
    if a % 2 == 0:
```

```
        print('10보다 큰 짝수')
```

```
    else:
```

```
        print('10보다 큰 홀수')
```

```
else:
```

```
    if a % 2 == 0:
```

```
        print('10 이하의 짝수')
```

```
    else:
```

```
        print('10 이하의 홀수')
```

a > 10 이 참인 경우 if a % 2 == 0:의 코드블록

a > 10 이 참인 경우 else: 의 코드블록

• 실행 결과

```
>ifif.py\n수를 입력하세요 : \n12\n10보다 큰 짝수
```

```
>ifif.py\n수를 입력하세요 : \n5\n10이하의 홀수
```

분기문 - if문

❖ 예제 : 06/ifand.py

```
print('수를 입력하세요 : ')\na = int(input())\n\nif a > 10 and a % 2 == 0:\n    print('10보다 큰 짝수')\nelif a > 10 and a % 2 != 0:\n    print('10보다 큰 홀수')\nelif a % 2 == 0 :\n    print('10이하의 짝수')\nelse :\n    print('10이하의 홀수')
```

• 실행 결과

```
> ifand.py\n수를 입력하세요 : \n1\n10이하의 홀수\n\n> ifand.py\n수를 입력하세요 : \n2\n10이하의 짝수
```



반복문 - while문

❖ 반복문

- 프로그램의 흐름을 되풀이하는 흐름 제어문.
- 루프문(Loop Statement)라고도 함.

❖ while문

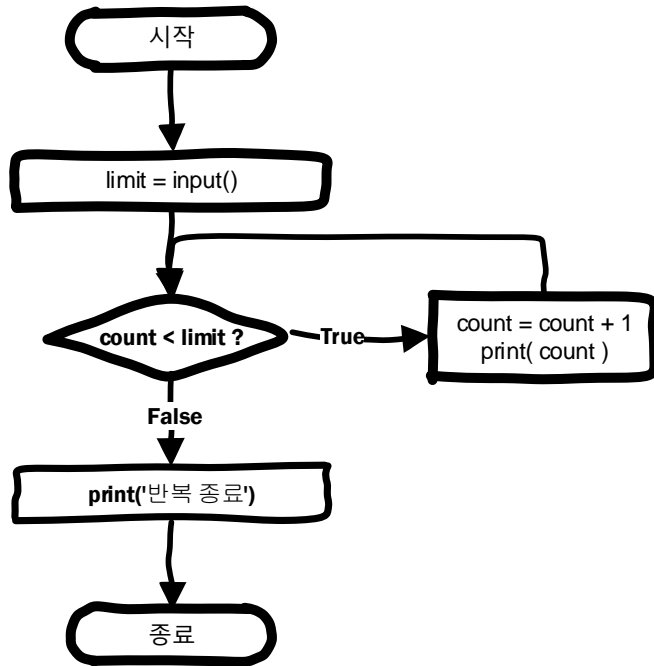
- while 키워드가 먼저 위치하고 그 다음에 조건이 위치.
- 조건 뒤에는 코드블록을 위치시키기 전에 콜론 : 이 위치.
- while이 영어로 “~하는 동안”이라는 뜻이므로 while 코드를 읽을 때는 “조건이 참인 동안”이라고 해석

```
while 조건:  
    코드블록
```



반복문 - while문

❖ 예제 : 06/while.py



```
print('몇 번 반복할까요? : ')
limit = int(input())

count = 0
while count < limit:
    count = count + 1
    print('{0}회 반복.'.format(count))

print('반복이 종료되었습니다.')
```

• 실행결과

```
>while.py
몇 번 반복할까요? :
5
1회 반복.
2회 반복.
3회 반복.
4회 반복.
5회 반복.
반복이 종료되었습니다.
```

반복문 - while문

❖ 무한 루프

- while문의 조건이 항상 참이 되는 루프

```
while True:  
    코드블록
```

❖ 예제 : 06/while_infinite.py

```
while True:  
    print('반복을 계속할까요? [예/아니오] :')  
    answer = input()  
  
    if answer == '예':  
        print('반복을 계속합니다.')  
    elif answer == '아니오':  
        break  
    else:  
        print('정상적인 답변이 아닙니다.')
```

break는 반복문의 실행을 취소

• 실행결과

```
>while_infinite.py  
반복을 계속할까요? [예/아니오] :  
예  
반복을 계속합니다.  
반복을 계속할까요? [예/아니오] :  
예  
반복을 계속합니다.  
반복을 계속할까요? [예/아니오] :  
yes  
정상적인 답변이 아닙니다.  
반복을 계속할까요? [예/아니오] :  
아니오  
반복이 종료되었습니다.
```

반복문 - for문

- ❖ for문은 조건을 평가하는 대신 순서열을 순회하다가 순서열의 끝에 도달하면 반복을 종료함.

```
for 반복변수 in 순서열:  
    코드블록
```

❖ 예제 : 06/for.py

```
for i in (1, 2, 3) :  
    print(i)
```

변수 i에는 매 반복마다 튜플 (1, 2, 3)의 요소들이 차례대로 복사됩니다. 튜플의 길이는 3이므로 이 for문은 3번 반복을 수행합니다.

- 실행 결과

```
>for.py  
1  
2  
3
```



반복문 - for문

❖ 예제 : 06/for_list.py

```
for s in ['뇌를', '자극하는', '파이썬']:  
    print(s)
```

- 실행 결과

```
>for_list.py  
뇌를  
자극하는  
파이썬
```

❖ 예제 : 06/for_string.py

```
for s in '뇌를 자극하는 파이썬':  
    print(s)
```

- 실행 결과

```
>for_string.py  
뇌  
를  
  
자  
극  
하  
는  
  
파  
이  
썬
```


반복문 - for문

❖ 실습 1 (range() 함수)

```
>>> for i in range(0, 5, 1):  
    print(i)
```

```
0  
1  
2  
3  
4
```

시작값

멈춤값

연속하는 두수의 차

```
>>> for i in range(0, 10, 2):  
    print(i)
```

```
0  
2  
4  
6  
8
```

❖ 실습 2 (range() 함수)

```
>>> for i in range(0, 5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

반복문 - for문

❖ 실습 3 (range() 함수)

```
>>> for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

❖ 예제 : 06/forfor.py

```
for i in range(1, 6):  
    for j in range(i):  
        print( "*", end = " ", )  
  
    print()
```

바깥의 for문에서 입력한 반복변수 i가 멈춤값입니다.

end = ""을 매개변수로 입력하면 줄바꿈을 출력하지 않습니다.

아무것도 입력하지 않으면 줄바꿈만 출력합니다.

• 실행 결과:

```
>for for .py  
*  
**  
***  
****  
*****
```

반복문 - for문

❖ 실습 4 (딕셔너리 요소 순회)

```
>>> dic = {'애플': 'www.apple.com',  
          '파이썬': 'www.python.org',  
          '마이크로소프트': 'www.microsoft.com'}  
>>> for k, v in dic.items():  
    print("{0} : {1}".format(k, v))
```

튜플 언패킹을 통해 k와 v에는 각 요소의 키와 값이 할당됩니다.

파이썬 : www.python.org

애플 : www.apple.com

마이크로소프트 : www.microsoft.com



반복문 - continue와 break로 반복문 제어하기

❖ continue

- 반복문이 실행하는 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 건너가도록 흐름을 조정

❖ 예제 : 06/continue.py

```
for i in range(10):  
    if i % 2 == 1:  
        continue  
  
    print(i)
```

i가 홀수일 때 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 바로 건너갑니다.

continue가 실행되는 경우에는 이 코드는 실행되지 않습니다.

• 실행 결과

```
>continue.py  
0  
2  
4  
6  
8
```



반복문 - continue와 break로 반복문 제어하기

❖ break

- 루프를 중단시키는 기능 수행

❖ 예제 : 06/break.py

```
i = 0
while(True):
    i = i+1
    if i == 1000:
        print('i가 {0}이 되었습니다. 반복문을 중단합니다.'.format(i))
        break

print(i)
```

- 실행 결과

```
>break.py
0
1
2
...
997
998
999
i가 1000이 되었습니다. 반복문을 중단합니다.
```



Thank You !

파이썬 프로그래밍