실전코딩

01. 파이썬 프로그래밍 리뷰 (1/2)
- 기본 문법

강원대학교 컴퓨터공학과 박치현

파이썬 프로그래밍 리뷰

1. 기본 문법

2. 함수, 객체지향프로그래밍

3. 자료구조와 알고리즘

Outline

파이썬의 구조

타입, 값, 변수 및 연산자

- 변수, 상수와 리터럴
- 자료형
- 형변환
- 연산자

조건문

- 조건문
- if, else 문
- 중첩 조건문

반복문

- for loop
- while loop
- 중첩 반복문

파이썬의 특징

생산성이 뛰어나다.

간결하면서도 효율적인 프로그램을 빠르게 작성

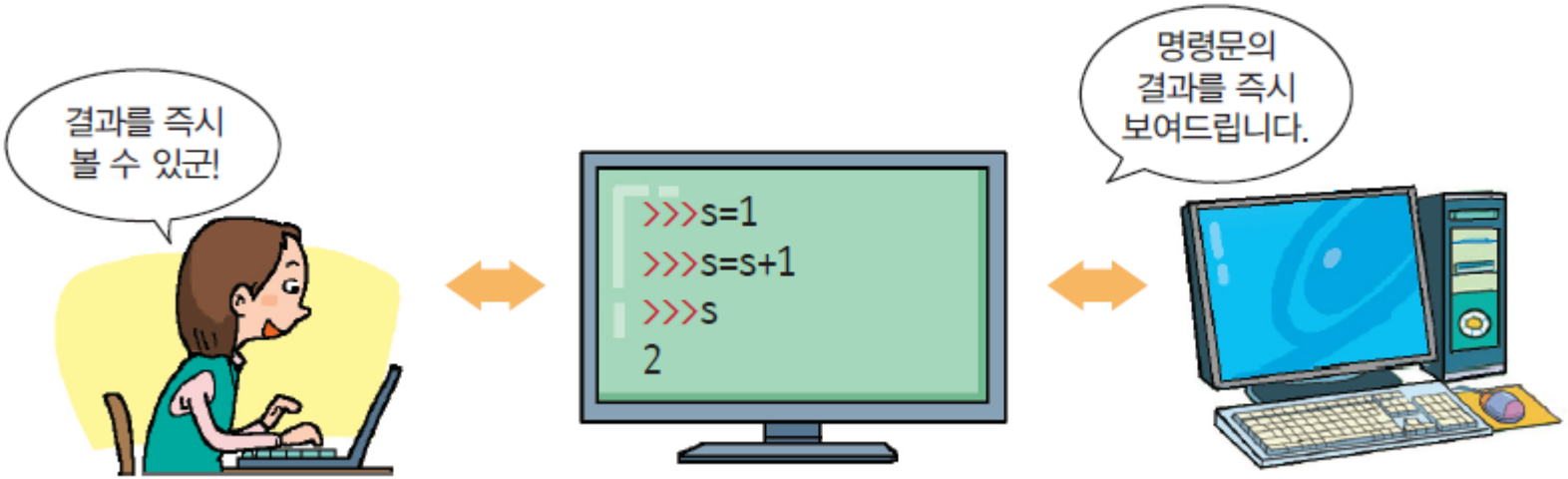
C 언어	파이썬

#include <stdio.h>
 	
int main(void) {
  printf("Hello World! \n");
  return 0;
}
	print("Hello World!")

천인국 저 '파이썬 Express' 참고

파이썬의 특징

인터프리터 언어: 파이썬 프로그래머는 자신이 작성한 명령문의 결과를 즉시 볼 수 있기 때문에 초보 프로그래머한테는 아주 바람직



파이썬의 특징

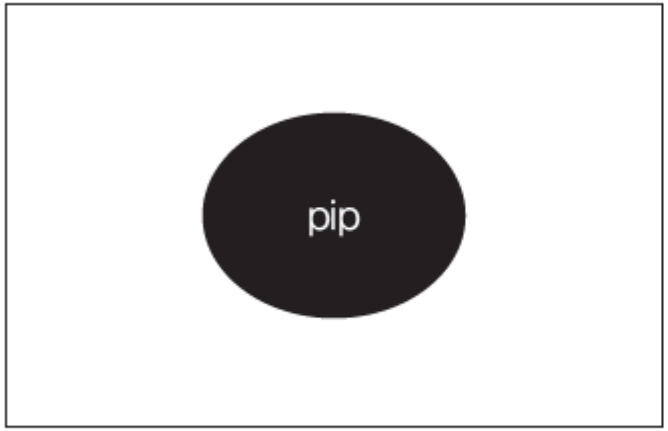
라이브러리가 풍부

라이브러리 설치가 쉽다.

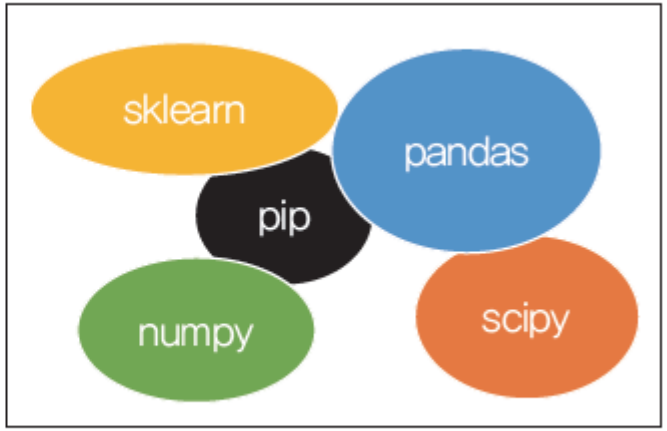
천인국 저 '파이썬 Express' 참고

아나콘다

인기 있는 라이브러리가 거의 모두 포함된 배포판



파이썬



아나콘다

천인국 저 '파이썬 Express' 참고

파이썬의 구조

파이썬의 구조

변수 상수

```
a = 8

b = 3

c = a + b

c = a * b
```

연산자

파이썬의 구조


```
def add(a, b):
    return a+b

def func1(a, b):
    if (a > b):
        for i in range(11):
            c = c + i
    return c
```

파이썬의 구조


```
def add(a, b):
    return a+b
```

```
def func1(a, b):
    if (a > b):
        for i in range(11):
            c = c + i
    return c
```

조건문

파이썬의 구조


```
def add(a, b):
    return a+b
```

```
def func1(a, b):
    if (a > b):
        for i in range(11):
            c = c + i
    return c
```

반복문

파이썬의 구조


```
class Calculator:
    a = 0
    b = 0
    def add( self ):
        return self.a+self.b
```

객체지향프로그래밍 (Object Oriented Programming) 특징

-캡슐화 (Encapsulation)

-상속성 (Inheritance)

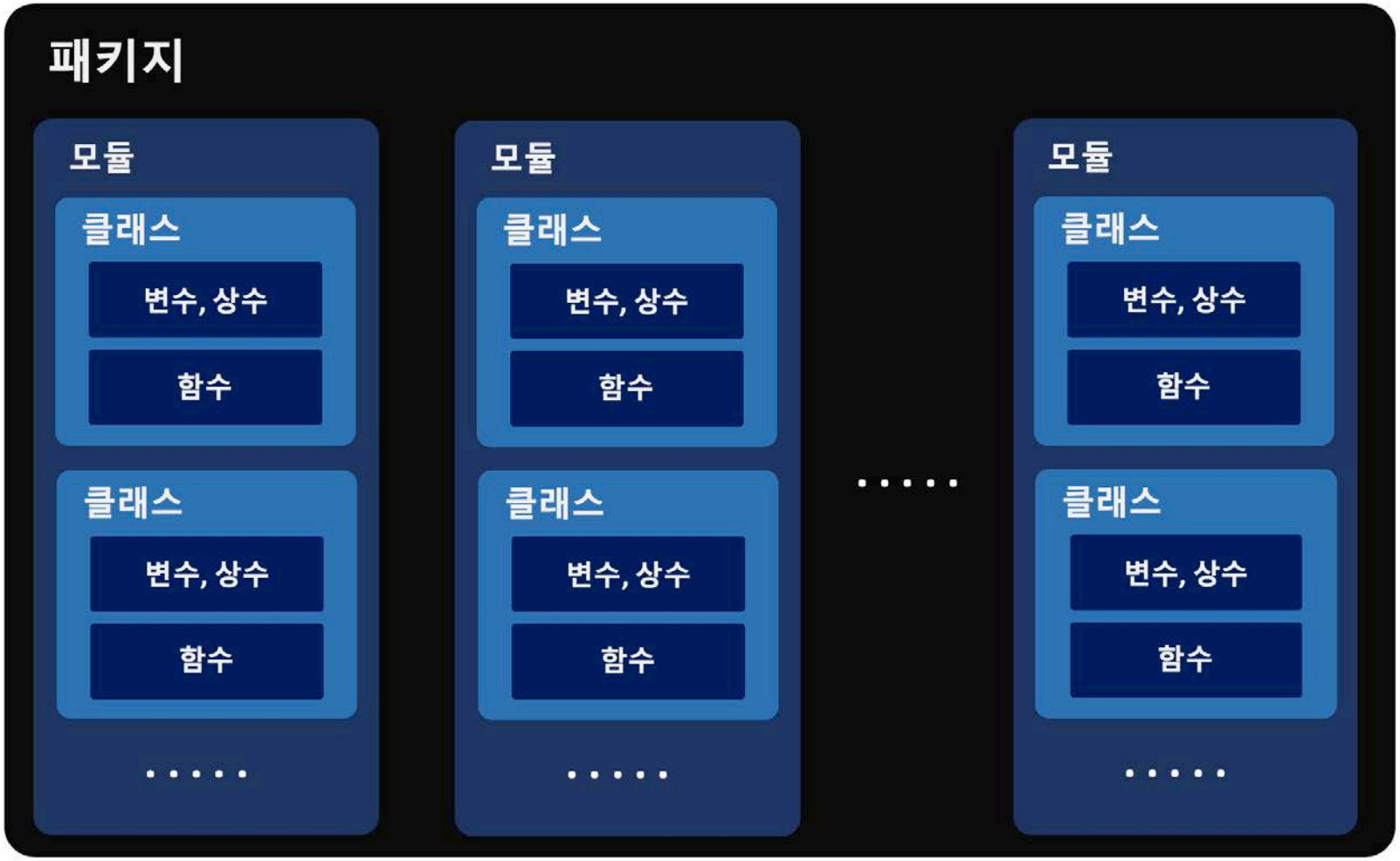
-다형성 (Polymorphism)

파이썬의 구조


```
import math:


math.pi #3.141592653589793
math.pow(2, 3) #8.0
math.factorial(5) #120
```

- 연관된 기능들의 집합
- 데이터, 함수, 클래스의 집합
- 재사용 용이

파이썬의 구조


패키지
모듈
클래스
변수, 상수
함수
클래스
변수, 상수
함수
.....
모듈
클래스
변수, 상수
함수
클래스
변수, 상수
함수
.....
모듈
클래스
변수, 상수
함수
클래스
변수, 상수
함수
.....

변수, 상수, 리터럴

- 변수

  프로그램에서 일시적으로 데이터를 저장하는 공간

- 상수

  프로그램의 실행 시작부터 끝날 때까지 값이 변하지 않는 공간
  (프로그래머나 시스템에 의해 미리 정해져 있는 값.)

  예. 수학/물리에서 파이값?
  PI = 3.14

- 리터럴

  값 그 자체를 의미
  (1, 2, 3, 4 처럼 변하지 않는 고유의 값)

  예.
  a = 0b011 # 이진법 표기 시 앞에 0b를 붙여준다
  print(a)

  출력) 3

키워드

고유한 의미를 갖는 예약된 단어

이미 예약되어 다른 용도로 사용할 수 없는 문자열

파이썬의 키워드

False	None	True	and	as	Assert	async
await	break	class	continue	def	del	elif
else	except	finally	for	from	global	if
import	in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with	yield

```
import keyword
kwlist = keyword.kwlist

for kw in kwlist:
    print(kw)
```


식별자

키워드 외에 프로그램에서 사용자가 필요에 따라 이름을 만들어 사용하는 단어

변수, 상수, 함수, 클래스 등의 구분을 위해 사용하는 이름

식별자 작성 규칙

- 영문 대/소문자(A~Z, a~z), 숫자(0~9), 밑줄(_)을 포함한 문자로만 구성
- 사용불가 : 식별자, 숫자로 시작하는 문자열, 특수기호(!, @, #, $, % 등)가 포함된 문자열

c.f.) 사용불가 식별자를 사용하는 경우 **:** 문법오류

```
SyntaxError: invalid syntax
```


변수(variable)

변수

- 프로그램에서 일시적으로 데이터를 저장하는 공간
- 식별자 작성 규칙에 따라 변수 이름 지정
- 입력된 데이터의 재사용 용이
- 메모리에 저장

변수 선언 및 할당

- 식별자 작성 규칙에 따라 생성된 이름과 대입연산자 (=) 를 이용
- e.g.)

```
var = 3
print(var)
```

  - **var**은 식별자 작성 규칙에 따라 생성된 변수의 이름
  - 대입연산자**(=)**를 통해 **3**이라는 정수가 **var**에 할당
  - 대입연산자**(=)**는 우변의 값을 좌변에 할당하라는 의미

변수(variable)

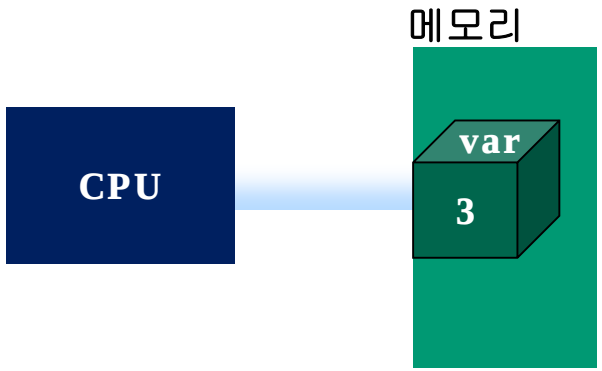
변수

- 프로그램에서 일시적으로 데이터를 저장하는 공간
- 식별자 작성 규칙에 따라 변수 이름 지정
- 입력된 데이터의 재사용 용이
- 메모리에 저장

변수 선언 및 할당

- 식별자 작성 규칙에 따라 생성된 이름과 대입연산자 (=) 를 이용
- e.g.)

```
var = 3
print(var)
```



변수(variable)

변수 선언 및 할당

- 식별자 작성 규칙에 따라 생성된 이름과 대입연산자 (=) 를 이용
- e.g.)

```
var1 = 1

var2 = 2

var3 = 3
```

```
print(var1)
print(var2)
print(var3)
```

=

```
var1, var2, var3 = 1, 2, 3
```

```
print(var1)
print(var2)
print(var3)
```


상수(constant)

프로그램의 실행 시작부터 끝까지 값이 변하지 않는 자료

- e.g.)

```
PI = 3.14

THRESHOLD = 0.8
```

c.f.) 대문자로 구분

상수(constant)

프로그램의 실행 시작부터 끝까지 값이 변하지 않는 자료

- e.g.)

```
PI = 3.14
```
→ 원주율 (상수)

```
radius = 10
```
→ 반지름 (변수)

```
area = radius * radius * PI
print(area)
```


리터럴(literal)

값 자체를 의미

- 숫자
- 문자
- 논리값
- 특수값
- 컬렉션

리터럴(literal)

값 자체를 의미

- 숫자 : **정수**, 실수, 복소수
- 문자
- 논리값
- 특수값
- 컬렉션

```
int_var = 10

print(int_var)
```

리터럴(literal)

값 자체를 의미

- 숫자 : 정수, **실수**, 복소수
- 문자
- 논리값
- 특수값
- 컬렉션

```
float_var1 = 83.5
print(float_var1) # float_var1의 결과 : 83.5

float_var2 = 0.835e2
print(float_var2) # float_var2의 결과 : 83.5
```


리터럴(literal)

값 자체를 의미

- 숫자
- 문자
- 논리값
- 특수값
- 컬렉션

```
str_var1 = "string"

print(str_var1)

str_var2 = 'string'

print(str_var2)
```

```
str_var1 = "string"

print(str_var1) # str_var1의 결과 : string

str_var2 = 'string'

print(str_var2) # str_var2의 결과 : string
```

리터럴(literal)

값 자체를 의미

- 숫자
- 문자
- 논리값 **True , False**
- 특수값 **None**
- 컬렉션 **list, tuple, dictionary** 등

자료형(data type)

프로그램에서 사용되는 자료 형태, 데이터를 식별하는 분류

- 숫자
- 문자

자료형(data type)

프로그램에서 사용되는 자료 형태, 데이터를 식별하는 분류

- 숫자
- 문자

```
int_var = 10  #우변의 형식에 따라 변수의 자료형 결정

print(int_var)

print(type(int_var)) #type(int_var)의 결과 : <class 'int'>
```

c.f.) type(변수이름) → 변수이름의 자료형을 return

int_var 는 정수형 **(int)** 자료형

자료형(data type)

프로그램에서 사용되는 자료 형태, 데이터를 식별하는 분류

- 숫자
- 문자

```
bin_var = 0b10
print(type(bin_var)) #type(bin_var)의 결과 : <class 'int'>
oct_var = 0o10
print(type(oct_var)) #type(oct_var)의 결과 : <class 'int'>
hex_var = 0x10
print(type(hex_var)) #type(hex_var)의 결과 : <class 'int'>
```


자료형(data type)

프로그램에서 사용되는 자료 형태, 데이터를 식별하는 분류

- 숫자
- 문자

```
float_var1 = 83.5
print(type(float_var1)) #type(bin_var)의 결과 : <class 'float'>
float_var2 = 0.835e2
print(type(float_var2)) #type(oct_var)의 결과 : <class 'float'>
com_var1 = 8j
print(type(com_var1))    #type(hex_var)의 결과 : <class 'complex'>
com_var2 = 15 + 8j
print(type(com_var2))    #type(hex_var)의 결과 : <class 'complex'>
```


자료형(data type)

프로그램에서 사용되는 자료 형태, 데이터를 식별하는 분류

- 숫자
- 문자

```
str_var1 = "string"

print(type(str_var1))  #type(str_var1)의 결과 : <class 'str'>
```

str_var1는 문자형 **(str)** 자료형

형변환(casting)

현재의 자료형을 (리터럴이나 변수) 다른 타입으로 변환하는 것

- e.g.) 정수형 ↹ 실수형

```
var1 = 3
var2 = 8.3

print(var1) #var1의 결과 : 3
print(var2) #var2의 결과 : 8.3

print(type(var1)) #type(var1)의 결과 : <class 'int'>
print(type(var2)) #type(var2)의 결과 : <class 'float'>
```


형변환(casting)

현재의 자료형을 (리터럴이나 변수) 다른 타입으로 변환하는 것

- e.g.) 정수형 ↹ 실수형

```
var1 = 3
var2 = 8.3

var1 = float(var1)
var2 = int(var2)

print(var1) #var1의 결과 : 3.0
print(var2) #var2의 결과 : 8

print(type(var1)) #type(var1)의 결과 : <class 'float'>
print(type(var2)) #type(var2)의 결과 : <class 'int'>
```

c.f.) **float(**변수이름**)** → 변수이름의 자료형을 **float**로 변환
int(변수이름**)** → 변수이름의 자료형을 **int**로 변환

형변환(casting)

현재의 자료형을 (리터럴이나 변수) 다른 타입으로 변환하는 것

- e.g.) 정수형 ↹ 문자형

```
var3 = 5
var4 = '15'

print(var3) #var3의 결과 : 5
print(var4) #var4의 결과 : 15

print(type(var3)) #type(var3)의 결과 : <class 'int'>
print(type(var4)) #type(var4)의 결과 : <class 'str'>

var3 = str(var3)   c.f.) str(변수이름) → 변수이름의 자료형을 str로 변환
var4 = int(var4)

print(var3) #var3의 결과 : 5
print(var4) #var4의 결과 : 15

print(type(var3)) #type(var3)의 결과 : <class 'str'>
print(type(var4)) #type(var4)의 결과 : <class 'int'>
```


연산자(operations)

연산을 수행하는 기호

- 산술연산자
- 관계연산자
- 논리연산자
- 삼항연산자
- 비트연산자
- 복합대입연산자
- 맴버연산자
- 식별연산자

연산자(operations)

연산을 수행하는 기호

- 산술연산자

연산자	의미
+	더하기
-	빼기
*	곱하기
/	나누기
%	나머지
**	제곱
//	몫

연산자(operations)

연산을 수행하는 기호

- 산술연산자

연산자	의미
+	더하기
-	빼기
*	곱하기
/	나누기
%	나머지
//	몫
**	제곱

```
var1 = 8
var2 = 3

print(var1+var2) #var1+var2의 결과 : 11

print(var1-var2) #var1-var2의 결과 : 5

print(var1*var2) #var1*var2의 결과 : 24

print(var1/var2) #var1/var2의 결과 : 2.66666...

print(var1%var2)  #var1%var2의 결과 : 2

print(var1//var2) #var1//var2의 결과 : 2

print(var1**var2) #var1**var2의 결과 : 512
```


연산자(operations)

연산을 수행하는 기호

- 관계연산자

연산자	의미
==	같다
!=	같지 않다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다

연산자(operations)

연산을 수행하는 기호

- 관계연산자 : 결과는 논리값 (True or False)

연산자	의미
==	같다
!=	같지 않다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다

```
var1 = 8
var2 = 3

print(var1 == var2) #var1 == var2의 결과 : False

print(var1 != var2) #var1 != var2의 결과 : True

print(var1 > var2)  #var1 > var2의 결과 : True

print(var1 >= var2) #var1 >= var2의 결과 : True

print(var1 < var2)  #var1 < var2의 결과 : False

print(var1 <= var2) #var1 <= var2의 결과 : False
```


연산자(operations)

연산을 수행하는 기호

- 논리연산자

연산자	의미
and	논리곱
or	논리합
not	부정

c.f.) Truth table

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

```
var1 = True
var2 = True

print(var1 and var2) #결과 : True
print(var1 or var2)  #결과 : True
print(not var1)      #결과 : False

var1 = True
var2 = False

print(var1 and var2) #결과 : False
print(var1 or var2)  #결과 : True
print(not var1)      #결과 : False
```


연산자(operations)

연산을 수행하는 기호

- 논리연산자

연산자	의미
and	논리곱
or	논리합
not	부정

c.f.) Truth table

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

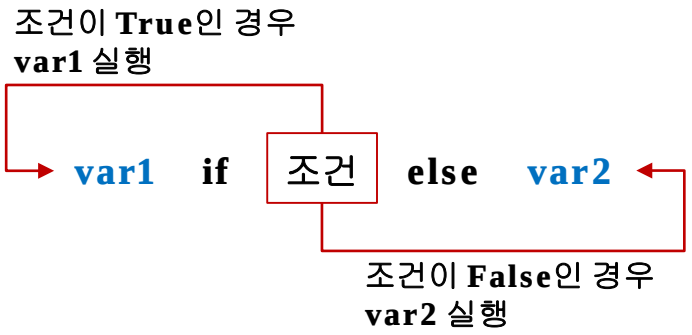
```
var1 = False
var2 = True

print(var1 and var2) #결과 : False
print(var1 or var2)  #결과 : True
print(not var1)      #결과 : True

var1 = False
var2 = False

print(var1 and var2) #결과 : False
print(var1 or var2)  #결과 : False
print(not var1)      #결과 : True
```


연산자(operations)

연산을 수행하는 기호

- 삼항연산자

var1 **if** 조건 **else** **var2**



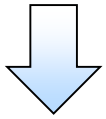
연산자(operations)

연산을 수행하는 기호

- 삼항연산자

var1 if 조건 else var2

```
var1 = 8
var2 = 3

print(var1 if var1 > var2 else var2)
```



조건이 **True**이므로
var1 실행

```
print(var1 if var1 > var2 else var2) #결과 : 8
```


연산자(operations)

연산을 수행하는 기호

- 비트연산자

연산자	의미
&	and 연산
|	or 연산
^	xor 연산
~	not 연산 (1의 보수)
<<	왼쪽 시프트
>>	오른쪽 시프트

c.f.) Truth table

a	b	a xor b
True	True	False
True	False	True
False	True	True
False	False	False

연산자(operations)

연산을 수행하는 기호

- 비트연산자

연산자	의미
&	and 연산
|	or 연산
^	xor 연산
~	not 연산
<<	왼쪽 시프트
>>	오른쪽 시프트

```
var1 = 8
var2 = 15

print(var1 & var2) #결과 : 8

print(var1 | var2) #결과 : 15

print(var1 ^ var2) #결과 : 7

print(~var1)       #결과 : -9

print(var1 << 2)   #결과 : 32

print(var1 >> 2)   #결과 : 2
```


연산자(operations)

연산을 수행하는 기호

- 비트연산자

연산자	의미
&	and 연산
|	or 연산
^	xor 연산
~	not 연산
<<	왼쪽 시프트
>>	오른쪽 시프트

```
var1 = 8
var2 = 15

print(var1 & var2)#결과 : 8

print(var1 | var2)#결과 : 15
```

var1 의 2진수 :	0	0	0	0	1	0	0	0
var2의 2진수 :	0	0	0	0	1	1	1	1
var1 & var2 :	0	0	0	0	1	0	0	0
var1 | var2 :	0	0	0	0	1	1	1	1

연산자(operations)

연산을 수행하는 기호

- 비트연산자

연산자	의미
&	and 연산
|	or 연산
^	xor 연산
~	not 연산
<<	왼쪽 시프트
>>	오른쪽 시프트

```
var1 = 8
var2 = 15
```

var1의 2진수 :

0	0	0	0	1	0	0	0

var2의 2진수 :

0	0	0	0	1	1	1	1

```
print(var1 ^ var2) #결과 : 7

print(~var1)          #결과 : -9
```

var1 ^ var2 :

0	0	0	0	0	1	1	1

~var1 = -(var+1)

연산자(operations)

연산을 수행하는 기호

- 비트연산자

연산자	의미
&	and 연산
|	or 연산
^	xor 연산
~	not 연산
<<	왼쪽 시프트
>>	오른쪽 시프트

```
var1 = 8
var2 = 15
```

var1 의 2진수 :	0	0	0	0	1	0	0	0
var1 << 2 :	0	0	1	0	0	0	0	0
var1 >> 2 :	0	0	0	0	0	0	1	0

```
print(var1 << 2)    #결과 : 32

print(var1 >> 2)    #결과 : 2
```

연산자(operations)

연산을 수행하는 기호

- 복합대입연산자

연산자	의미
v1 += v2	v1 = v1 + v2
v1 -= v2	v1 = v1 - v2
v1 *= v2	v1 = v1 * v2
v1 /= v2	v1 = v1 / v2
v1 %= v2	v1 = v1 % v2
v1 //= v2	v1 = v1 // v2
v1 **= v2	v1 = v1 ** v2
v1 &= v2	v1 = v1 & v2
v1 |= v2	v1 = v1 | v2
v1 ^= v2	v1 = v1 ^ v2
v1 <<= v2	v1 = v1 << v2
v1 >>= v2	v1 = v1 >> v2

연산자(operations)

연산을 수행하는 기호

- 복합대입연산자

연산자	의미
v1 += v2	v1 = v1 + v2
v1 -= v2	v1 = v1 - v2
v1 *= v2	v1 = v1 * v2
v1 /= v2	v1 = v1 / v2
v1 %= v2	v1 = v1 % v2
v1 //= v2	v1 = v1 // v2
v1 **= v2	v1 = v1 ** v2
v1 &= v2	v1 = v1 & v2
v1 |= v2	v1 = v1 | v2
v1 ^= v2	v1 = v1 ^ v2
v1 <<= v2	v1 = v1 << v2
v1 >>= v2	v1 = v1 >> v2

```
var1 = 8
var2 = 3

var1 += var2
print(var1) #결과 : 11

var1 = 8
var2 = 3

var1 = var1 + var2
print(var1) #결과 : 11
```


연산자(operations)

연산을 수행하는 기호

- 맴버연산자

연산자	의미
in	포함 검사
not in	비포함 검사

c.f.) list : 자료들의 모임을 표현하는 자료형
list1 = [1, 2, 3, 4, 5]

```
list1 = [1, 2, 3, 4, 5]

var1 = 'test'

var2 = 'example'

print(var1 if 3 in list1 else var2) #결과 : test
```

list1에 3이 포함되어 있으면 var1 실행
포함되어 있지 않으면 var2 실행

연산자(operations)

연산을 수행하는 기호

- 맴버연산자

연산자	의미
in	포함 검사
not in	비포함 검사

c.f.) list : 자료들의 모임을 표현하는 자료형
list1 = [1, 2, 3, 4, 5]

```
list1 = [1, 2, 3, 4, 5]

var1 = 'test'

var2 = 'example'

print(var1 if 3 not in list1 else var2) #결과 : example
```

list1에 3이 포함되어 있지 않으면 var1 실행
포함되어 있으면 var2 실행

연산자(operations)

연산을 수행하는 기호

- 식별연산자

연산자	의미
is	객체 혹은 값이 같으면 True
is not	객체 혹은 값이 같지 않으면 True

```
var1 = 'test'

var2 = 'example'

print(var1 if 10 is 10 else var2)      #결과 : test

print(var1 if 10 is not 10 else var2) #결과 : example
```


연산자(operations)

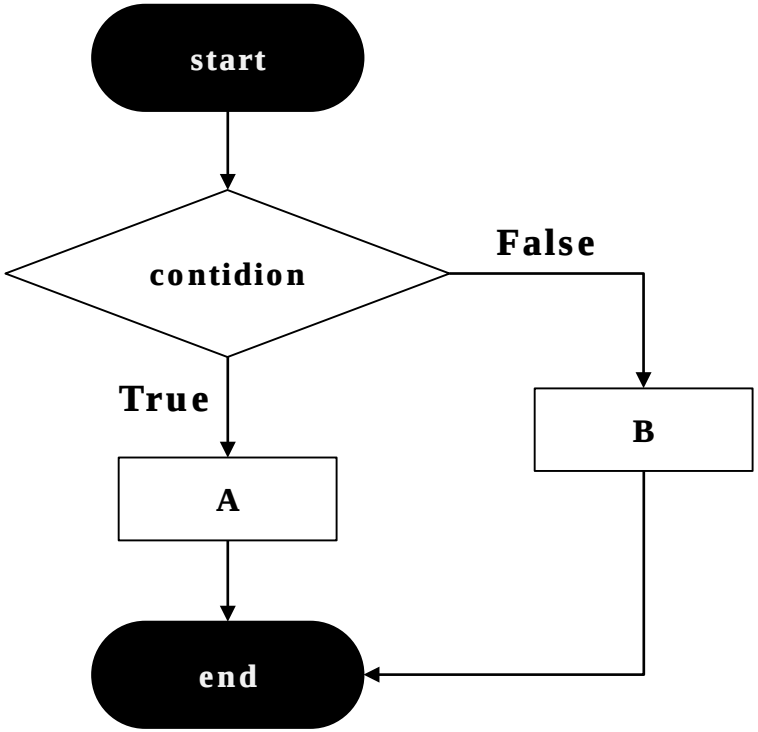
연산자 우선순위

연산자	의미
()	괄호
**	제곱
~, +, -	bitwise not, 부호
*, /, %, //	곱하기, 나누기, 나머지, 몫
+, -	덧셈, 뺄셈
>>, <<	비트 시프트
&	bitwise &
|, ^	bitwise or, bitwise xor
<, <=, >, >=, ==, !=	관계
=, %=, /=, //=, -=, +=, *=, **=	할당
is, is not	식별
in, not in	맴버
not, or, and	논리
if – else	삼항

조건문

- 조건문

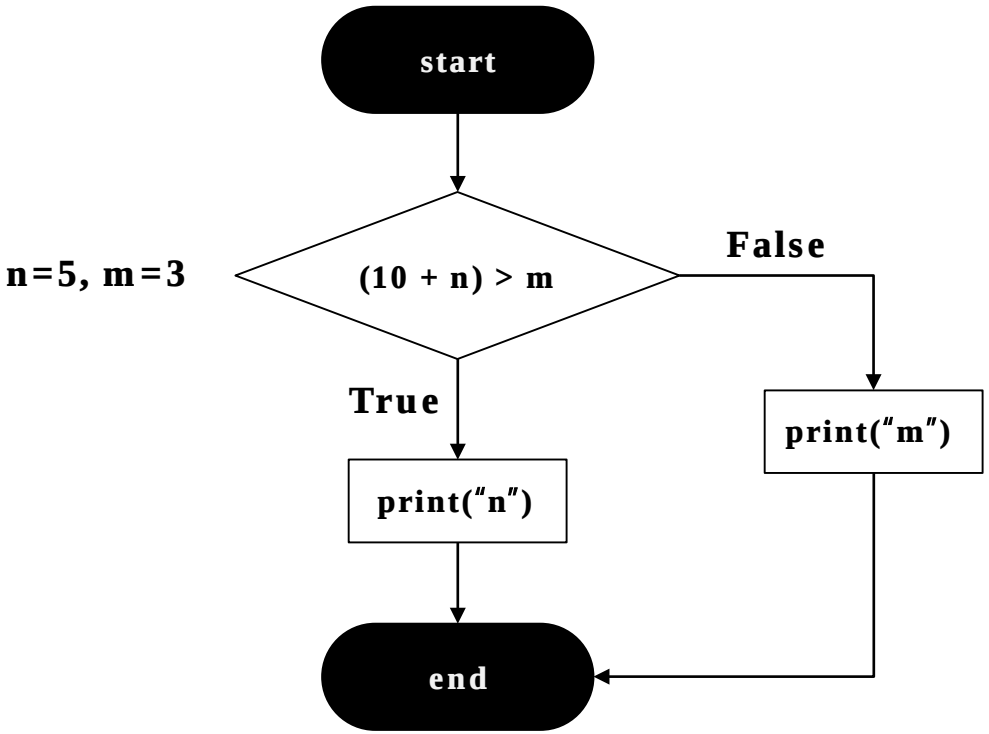
  명시한 조건의 **True** 혹은 **False**의 결과에 따라 프로그램 실행여부 **(**흐름**)** 를 결정하는 기능



조건문

- 조건문

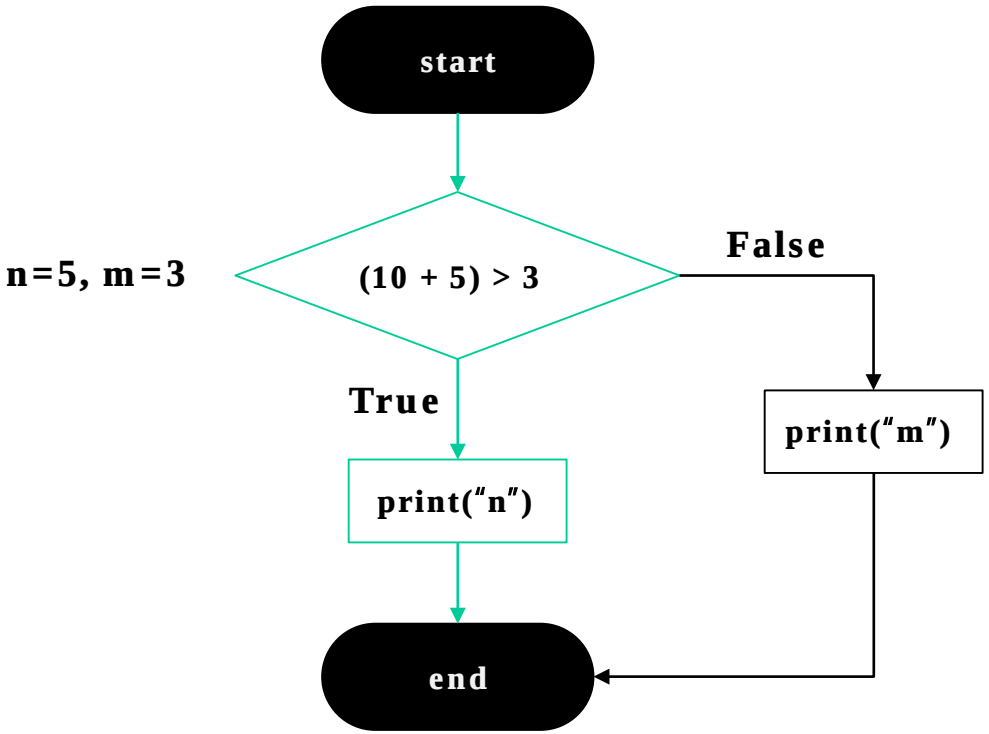
  명시한 조건의 **True** 혹은 **False**의 결과에 따라 프로그램 실행여부 **(**흐름**)** 를 결정하는 기능



조건문

- 조건문

  명시한 조건의 **True** 혹은 **False**의 결과에 따라 프로그램 실행여부 **(**흐름**)** 를 결정하는 기능



조건문

- 조건문

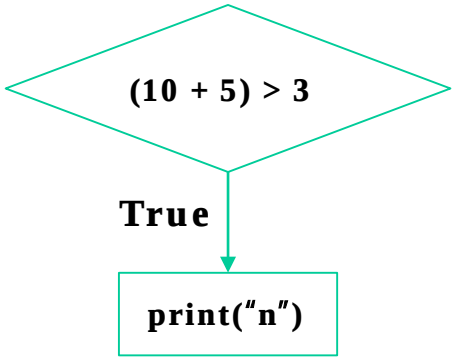
  명시한 조건의 **True** 혹은 **False**의 결과에 따라 프로그램 실행여부 **(**흐름**)** 를 결정하는 기능

Statement	Description
if	if문에 대한 조건을 명시
if, else	if part와 else part의 조건을 명시
중첩 조건문	if 문 내부에 if문을 사용하는 경우

if, else문

- **if문**

if (조건문) : 조건이 참인 경우 실행문장



1)

```
if(10 + 5 > 3): print("n") # 결과 : n
```

2)

```
if(10 + 5 > 3):
    print("n") # 결과 : n
```

c.f.) 조건문 결과가 논리리터럴인 경우 : **True**인 경우 실행, **False**인 경우 실행문 다음을 실행
조건문 결과가 논리리터럴이 아닌 **(value)** 경우 : **non-zero, non-null**이면 **True**로,
zero, null이면 **False**로 판단

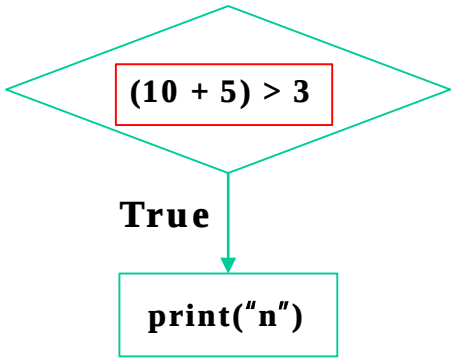
```
if(1):
    print("non-zero") # 결과 : non-zero

if(0):
    print("non-zero")
```

if, else문

- **if문**

if (조건식) : 조건이 참인 경우 실행문장

- 실행문이 **1**라인일 경우 **if** 문과 같은 라인에 작성가능



관계연산자 사용

1)

```
if(10 + 5 > 3): print("n") # 결과 : n
```

2)

```
if(10 + 5 > 3):
    print("n") # 결과 : n
```

if, else문

- **if문**

if (조건식) : 조건이 참인 경우 실행문장

- 조건식에 다양한 수식과 더불어 관계, 논리, 비트 연산 사용

- 다양한 관계연산자 사용 **examples**

```
var1 = 8
var2 = 8

if(var1 == var2): print("1. condition is TRUE.") # 결과 : 1. condition is TRUE.

if(var1 != var2): print("2. condition is TRUE.")

if(var1 > var2): print("3. condition is TRUE.")

if(var1 >= var2): print("4. condition is TRUE.") # 결과 : 4. condition is TRUE.

if(var1 < var2): print("5. condition is TRUE.")

if(var1 <= var2): print("6. condition is TRUE.") # 결과 : 6. condition is TRUE.
```


if, else문

- **if문**

  if (조건식) : 조건이 참인 경우 실행문장

  - 조건식에 다양한 수식과 더불어 관계, 논리, 비트 연산 사용

  - 다양한 논리연산자 사용 **examples**

```
var1 = True
var2 = False

if(var1 and var2): print("1. condition is TRUE.")

if(var1 or var2): print("2. condition is TRUE.")    # 결과 : 2. condition is TRUE.

if(not var2): print("3. condition is TRUE.")        # 결과 : 3. condition is TRUE.
```


if, else문

- **if문**

  if (조건식) : 조건이 참인 경우 실행문장

  - 조건식에 다양한 수식과 더불어 관계, 논리, 비트 연산 사용

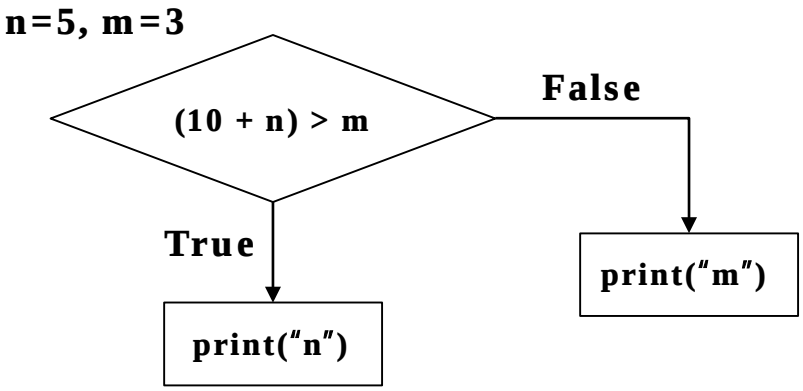
  - 다양한 비트연산자 사용 **examples**

```
var1 = True
var2 = False

if(var1 & var2): print("1. condition is TRUE.")

if(var1 | var2): print("2. condition is TRUE.")  # 결과 : 2. condition is TRUE.

if(var1 ^ var2): print("3. condition is TRUE.")  # 결과 : 3. condition is TRUE.

if(~var2): print("4. condition is TRUE.")        # 결과 : 4. condition is TRUE.
```


if, else문

- **if, else**

  if (조건식) :
  조건이 참인 경우 실행문장
  else:
  조건이 만족하지 않는 경우 실행문장

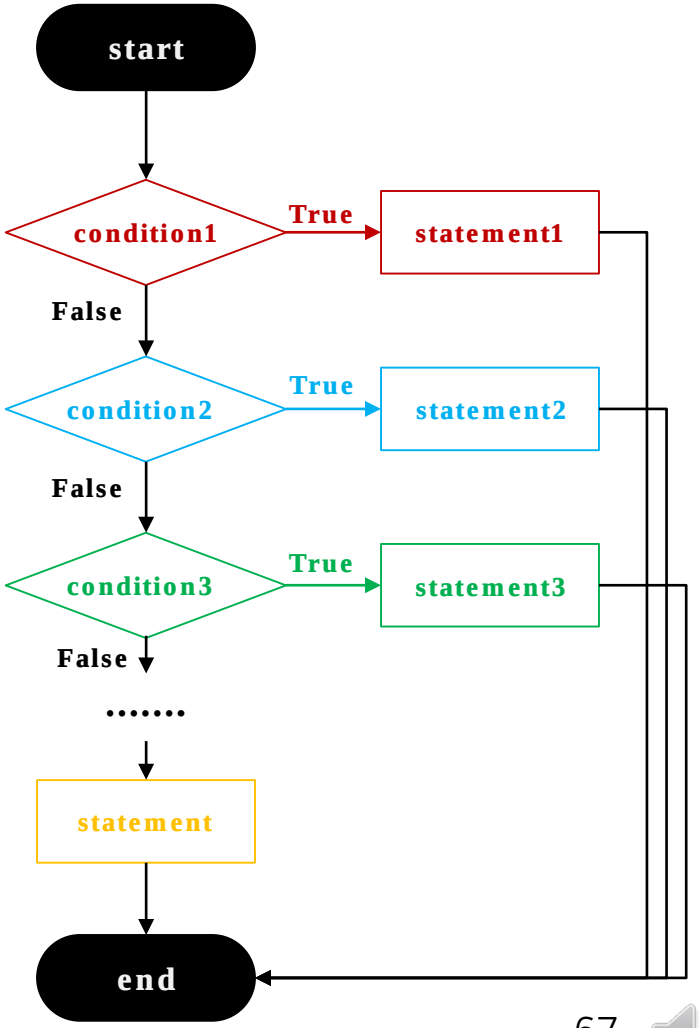



```
n=5
m=3

if((10+n) > m):
    print("n") # 결과 : n
else:
    print("m")
```

if, else문

- **if, else**

```
if (조건식1) :
        조건식1이 참인 경우 실행문장
elif(조건식2):
        조건식2가 참인 경우 실행문장
elif(조건식3):
        조건식3이 참인 경우 실행문장
.....
else:
        조건이 만족하지 않는 경우 실행문장
```



if, else문

- **if, else**

```
blood_type = "O"

print("test start: ")

if(blood_type == "A"):
    print("blood type is A")

elif(blood_type == "B"):
    print("blood type is B")

elif (blood_type == "AB"):
    print("blood type is AB")

elif (blood_type == "O"):
    print("blood type is O")

print("test end.")
```

실행결과

```
test start:
blood type is O
test end.
```


if, else문

- **if, else**

```
blood_type = "O"

print("test start: ")          → 실행

if(blood_type == "A"):         → False
    print("blood type is A")
elif(blood_type == "B"):       → False
    print("blood type is B")
elif (blood_type == "AB"):     → False
    print("blood type is AB")
elif (blood_type == "O"):      → True
    print("blood type is O")   → 실행

print("test end.")             → 실행
```


if, else문

- **if, else**

```
blood_type = "O"

print("test start: ")

if(blood_type == "O"):
    print("blood type is A")

elif(blood_type == "B"):
    print("blood type is B")

elif (blood_type == "AB"):
    print("blood type is AB")

elif (blood_type == "O"):
    print("blood type is O")

print("test end.")
```

실행

True

실행

실행

중첩 조건문

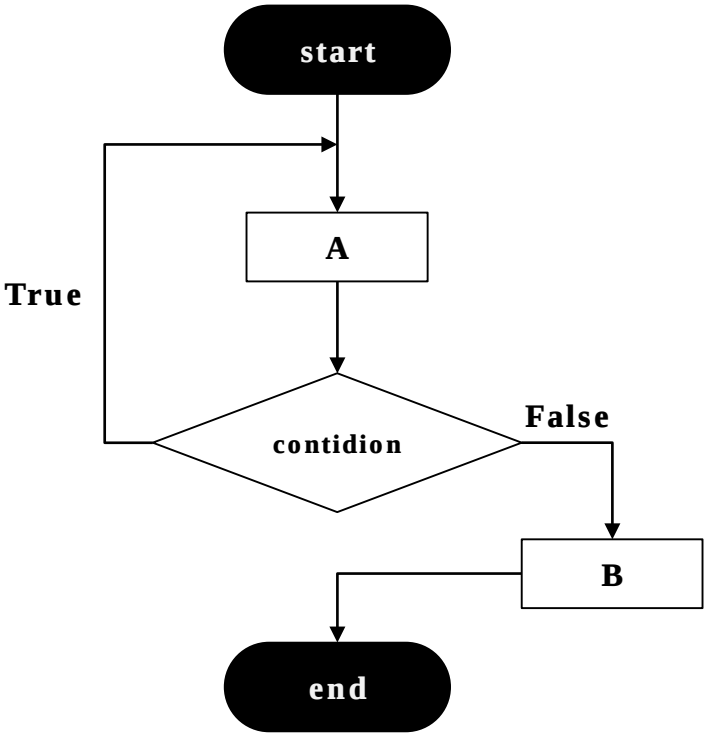
- 조건문 내부에 조건문

```
if (조건식1) :
        if(조건식1.1):
                조건식1.1이 참인 경우 실행문장
elif(조건식2):
        조건식2가 참인 경우 실행문장
elif(조건식3):
        if(조건식3.1):
                조건식3.1이 참인 경우 실행문장
        elif(조건식3.2):
                조건식3.2가 참인 경우 실행문장
        else:
                조건이 만족하지 않는 경우 실행문장
.......
else:
        조건이 만족하지 않는 경우 실행문장
```


반복문

- 반복문

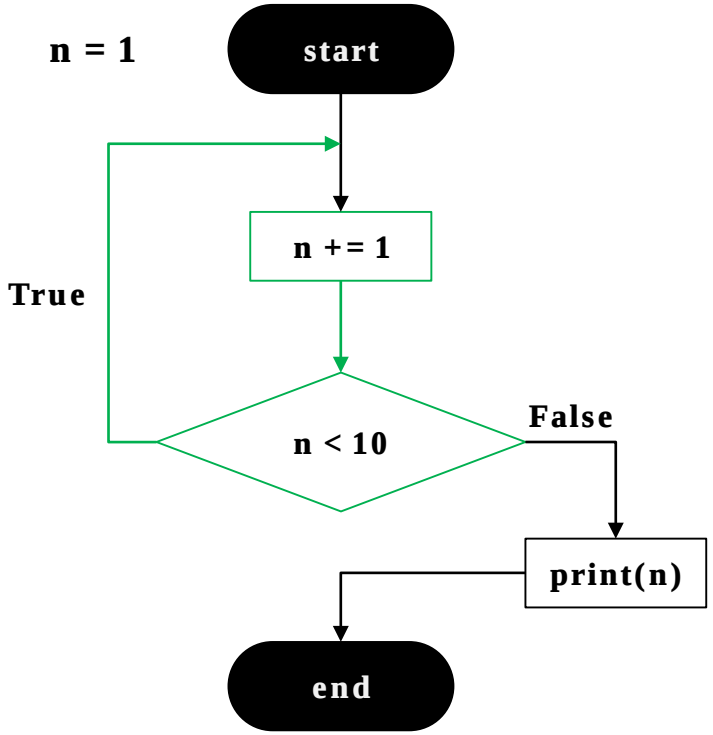
  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능



반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능



- while loop

while 조건문:
    조건이 참일동안 실행할 문장

c.f.) break – 반복문 탈출 키워드
continue – 반복문 처음으로 이동 키워드

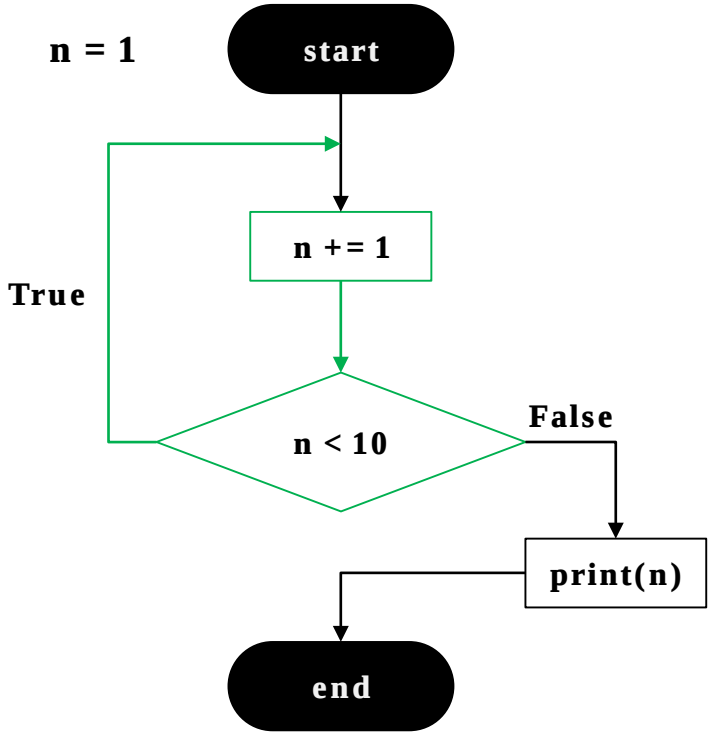
- for loop

for 변수명 **in** 데이터 집합명:
    반복 실행할 문장 (명령문)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능



- while loop

while 조건문:
조건이 참일동안 실행할 문장

```
n = 1
while(n < 10):
    n += 1
print(n)
```

반복문

- 반복문

특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

- while loop

while 조건문:
조건이 참일동안 실행할 문장

```
n = 1

while(n < 10):
    n += 1

print(n)
# 결과 : 10
```

1) n의 값 : 1 → n<10 → True → n+=1 → n=2

2) n의 값 : 2 → n<10 → True → n+=1 → n=3

3) n의 값 : 3 → n<10 → True → n+=1 → n=4

.....

8) n의 값 : 8 → n<10 → True → n+=1 → n=9

9) n의 값 : 9 → n<10 → True → n+=1 → n=10

10) n의 값 : 10 → n<10 → False → print(n)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

  - while loop

  while 조건문:
  조건이 참일동안 실행할 문장

```
n=1

while(n < 6):
    print("#")
    n += 1

print("test end.")
```

1) n의 값 : 1 → n<6 → True → print(“#”) → n+=1 → n=2

2) n의 값 : 2 → n<6 → True → print(“#”) → n+=1 → n=3

3) n의 값 : 3 → n<6 → True → print(“#”) → n+=1 → n=4

4) n의 값 : 4 → n<6 → True → print(“#”) → n+=1 → n=5

5) n의 값 : 5 → n<6 → True → print(“#”) → n+=1 → n=6

6) n의 값 : 6 → n<6 → False → print(“test end.”)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

  - while loop

  while 조건문:
  조건이 참일동안 실행할 문장

```
n=1

while(True):
    print("#")
    n += 1
    if(n > 5):
        break

print("test end.")
```

1) True → print(“#”) → n+=1 → n의 값 : 2 → n > 5 → False

2) True → print(“#”) → n+=1 → n의 값 : 3 → n > 5 → False

3) True → print(“#”) → n+=1 → n의 값 : 4 → n > 5 → False

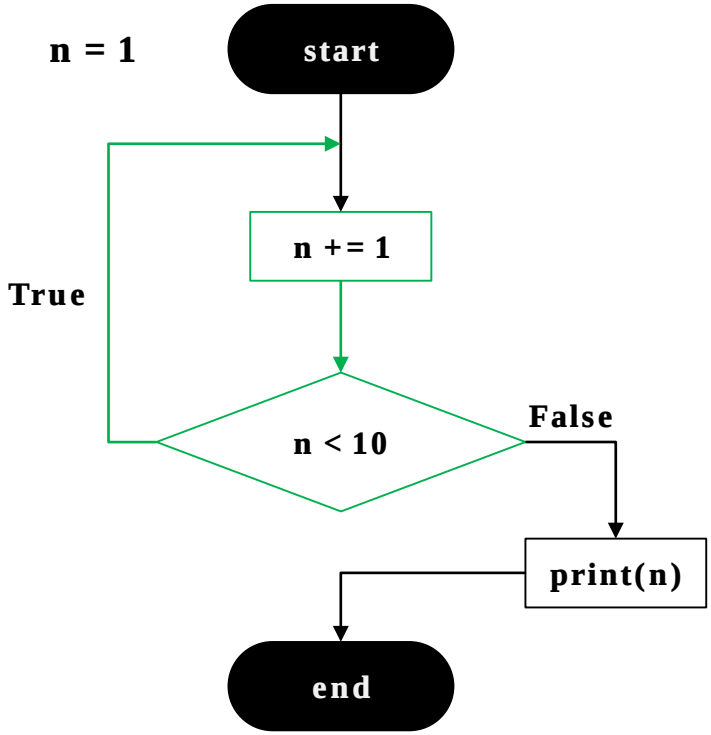
4) True → print(“#”) → n+=1 → n의 값 : 5 → n > 5 → False

5) True → print(“#”) → n+=1 → n의 값 : 6 → n > 5 → True → break → print(″test end.″)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능



- for loop

for 변수명 **in** 데이터 집합명:
    반복 실행할 문장 (명령문)

```
n = 1
for i in range(1, 10):
    n += 1
print(n)
```

c.f.) range(start, end) 함수
start 부터 **end-1**까지의 수 범위의 값을 리턴

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

  - for loop

  for 변수명 **in** 데이터 집합명:
  반복 실행할 문장 (명령문)

```
n = 1

for i in range(1, 10):
    n += 1

print(n)
```

1) i의 값 : 1 → n += 1 → n=2

2) i의 값 : 2 → n += 1 → n=3

3) i의 값 : 3 → n += 1 → n=4

.....

8) i의 값 : 7 → n += 1 → n=8

9) i의 값 : 8 → n += 1 → n=9

10) i의 값 : 9 → n += 1 → n=10 →print(n)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

  - for loop

  for 변수명 **in** 데이터 집합명:
  반복 실행할 문장 (명령문)

```
n = 1

for i in range(1, 6):
    print("#")

print("test end.")
```

1) i의 값 : 1 → print(“#”)

2) i의 값 : 2 → print(“#”)

3) i의 값 : 3 → print(“#”)

4) i의 값 : 4 → print(“#”)

5) i의 값 : 5 → print(“#”) → print(“test end.”)

반복문

- 반복문

  특정 실행문장 (명령문) 이 반복적으로 수행될 수 있도록 하는 기능

  - for loop

  for 변수명 **in** 데이터 집합명:
  반복 실행할 문장 (명령문)

```
n = 1

for i in range(1, 10):
    if(i == 6):
        break
    print("#")

print("test end.")
```

1) i의 값 : 1 → i == 6 → False → print(“#”)

2) i의 값 : 2 → i == 6 → False → print(“#”)

3) i의 값 : 3 → i == 6 → False → print(“#”)

4) i의 값 : 4 → i == 6 → False → print(“#”)

5) i의 값 : 5 → i == 6 → False → print(“#”)

6) i의 값 : 6 → i == 6 → True → break
→ print(“test end.”)

반복문

- 반복문

while loop vs. for loop

- while loop

while 조건문:
조건이 참일동안 실행할 문장

```
n = 1

while(n < 10):
    n += 1

print(n)
```

- for loop

for 변수명 **in** 데이터 집합명:
반복 실행할 문장 **(명령문)**

```
n = 1

for i in range(1, 10):
    n += 1

print(n)
```


반복문

- 반복문

while loop vs. for loop

- while loop

while 조건문:
조건이 참일동안 실행할 문장

```
n=1

while(n < 6):
    print("#")
    n += 1

print("test end.")
```

- for loop

for 변수명 **in** 데이터 집합명:
반복 실행할 문장 **(명령문)**

```
n = 1

for i in range(1, 6):
    print("#")

print("test end.")
```


중첩 반복문

- 반복문 내부에 반복문

- while loop

```
while 조건문:
    while 조건문:
        조건이 참일동안 실행할 문장
    while 조건문:
        조건이 참일동안 실행할 문장
.....
```

- for loop

```
for 변수명 in 데이터 집합명:
    for 변수명 in 데이터 집합명:
        반복 실행할 문장 (명령문)
    for 변수명 in 데이터 집합명:
        반복 실행할 문장 (명령문)
.....
```

```
while 조건문:
    for 변수명 in 데이터 집합명:
        반복 실행할 문장 (명령문)
    while 조건문:
        조건이 참일동안 실행할 문장
.....
```


중첩 반복문

- 반복문 내부에 반복문

```
n = 0
for i in range(1, 11):
    m = 1
    for j in range(1, 10):
        m += 1
    n += m
print(n)
```


중첩 반복문

- 반복문 내부에 반복문

```
n = 0

for i in range(1, 11):
    m = 1
    for j in range(1, 10):
        m += 1
    n += m

print(n)
```


j=1 일 때 m → 2
j=2 일 때 m → 3
j=3 일 때 m → 4
…
j=9 일 때 m → 10

1) i의 값 : 1 → m의 값 : 1 → 내부 for loop 실행 → m의 값 : 10 → n += m → n의 값 : 10

2) i의 값 : 2 → m의 값 : 1 → 내부 for loop 실행 → m의 값 : 10 → n += m → n의 값 : 20

3) i의 값 : 3 → m의 값 : 1 → 내부 for loop 실행 → m의 값 : 10 → n += m → n의 값 : 30

.....

9) i의 값 : 9 → m의 값 : 1 → 내부 for loop 실행 → m의 값 : 10 → n += m → n의 값 : 90

10) i의 값 : 10 → m의 값 : 1 → 내부 for loop 실행 → m의 값 : 10 → n += m → n의 값 : 100
→ print(n)

퀴즈 (O|X)

1. x+=y 처럼 대입 연산자와 다른 연산자를 합쳐 놓은 연산자를 '복합 대입 연산자'라고 한다.

2. for문에서 사용되는 range(start, end) 함수는 start 부터 end까지의 수 범위의 값을 리턴한다. 예를 들어 range(1,6)이면 for문 내부 명령문은 6번 반복 수행 된다.