Elixirr

파이썬 기본

인선미

2029.01

파이썬 시작하기

파이썬

- ✓ 플랫폼 독립성: 동일한 코드를 윈도우, 맥, 리눅스 등 다양한 운영체제에서 실행가능
- ✓ 동적 타이핑: 변수의 데이터 타입을 미리 선언하지 않음
- ✓ 멀티 패러다임 언어: 객체지향 프로그래밍, 절차적 함수형 프로그래밍
- ✓ 인터프리터 언어: 컴파일 과정 없이 코드를 바로 실행가능. 개발 주기 빠름
- ✓ 쉬운 확장성: C, C++, Java 등 다른 언어와의 통합 용이

파이썬의 강점

- ✓ 간결하고 유연한 문법
- ✓ 풍부한 라이브러리와 프레임워크
- ✓ 방대한 사용자 커뮤니티

파이썬의 특징

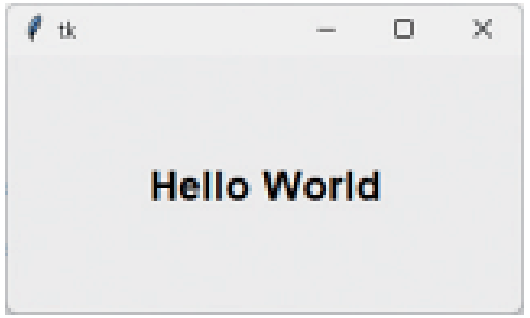
```
languages = ['python', 'perl', 'c', 'java']

for lang in languages:
    if lang in ['python', 'perl']:
        print("%6s need interpreter" % lang)
    elif lang in ['c', 'java']:
        print("%6s need compiler" % lang)
    else:
        print("should not reach here")
```

```
 python need interpreter
   perl need interpreter
      c need compiler
   java need compiler
```

파이썬 적용 분야

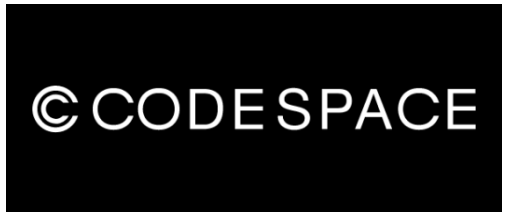
- ✓ 웹 프로그래밍
- ✓ 인공지능과 머신러닝
- ✓ 수치 연산 프로그래밍
- ✓ 데이터 분석
- ✓ 데이터베이스 프로그래밍
- ✓ 시스템 유틸리티 제작하기
- ✓ GUI 프로그래밍
- ✓ C/C++와 결합하기
- ✓ 사물 인터넷

NumPy



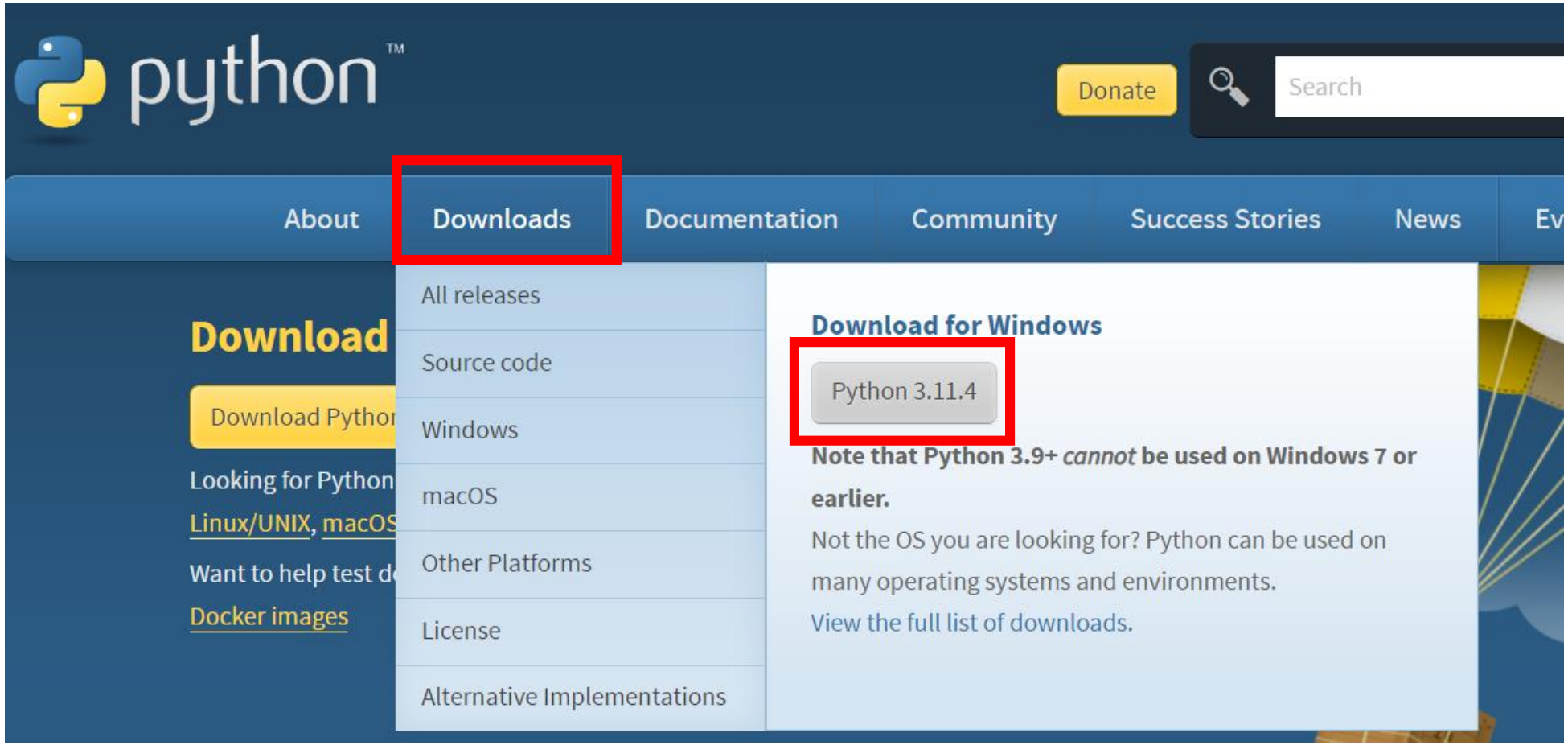
pandas

개발 환경

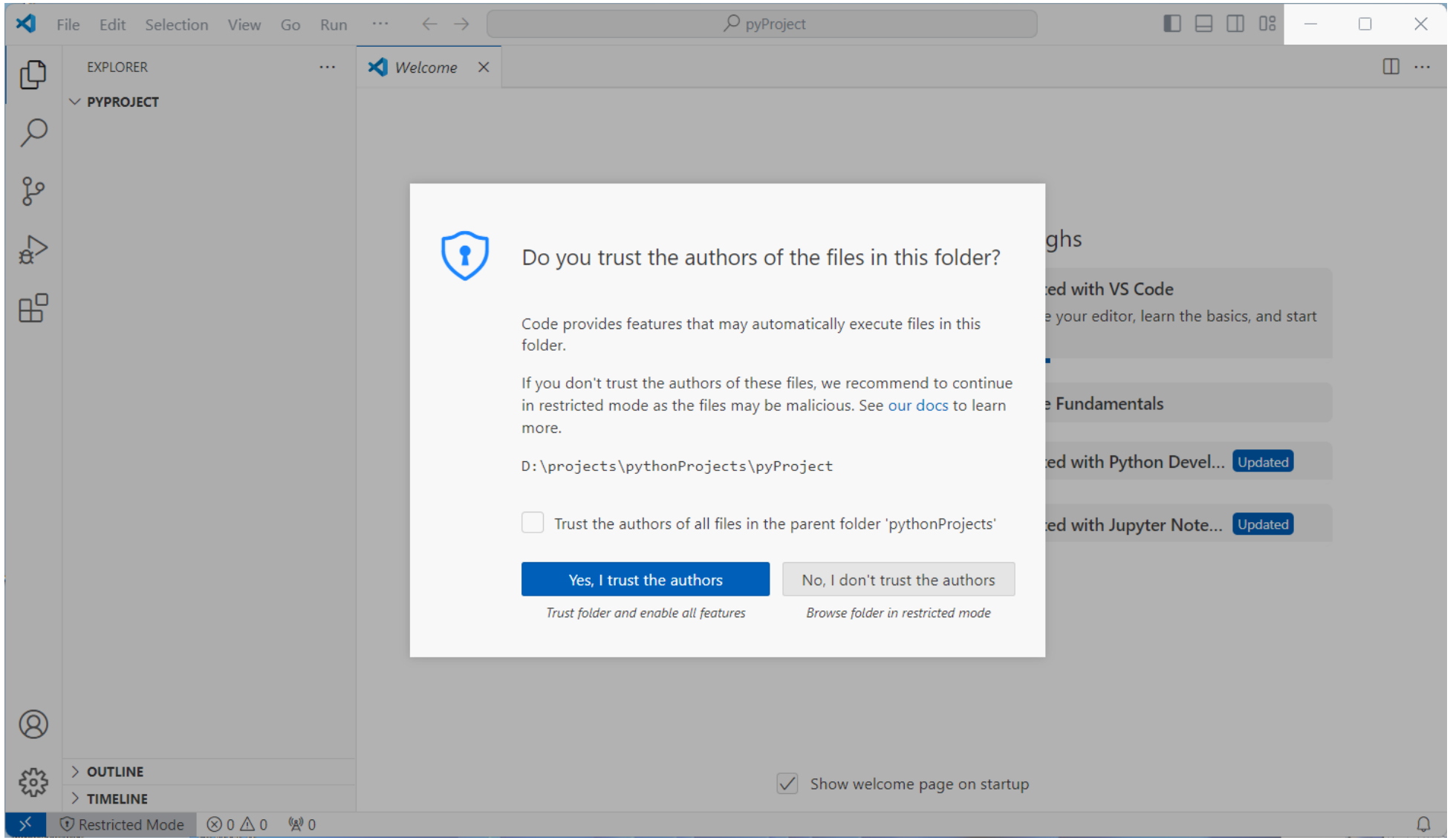


파이썬 설치

✓ http://www.python.org/에 접속

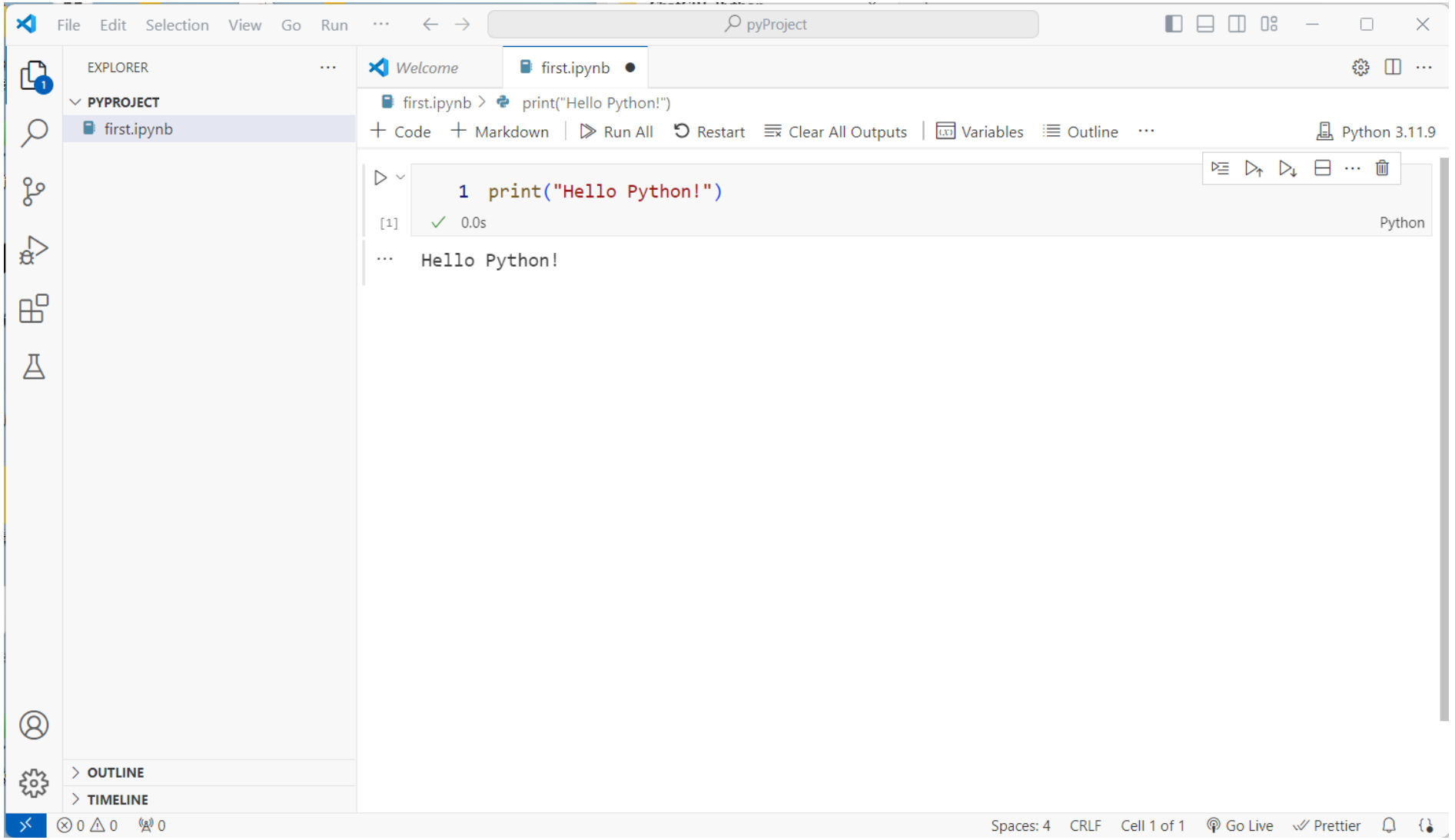


개발환경 설정



Elixirr

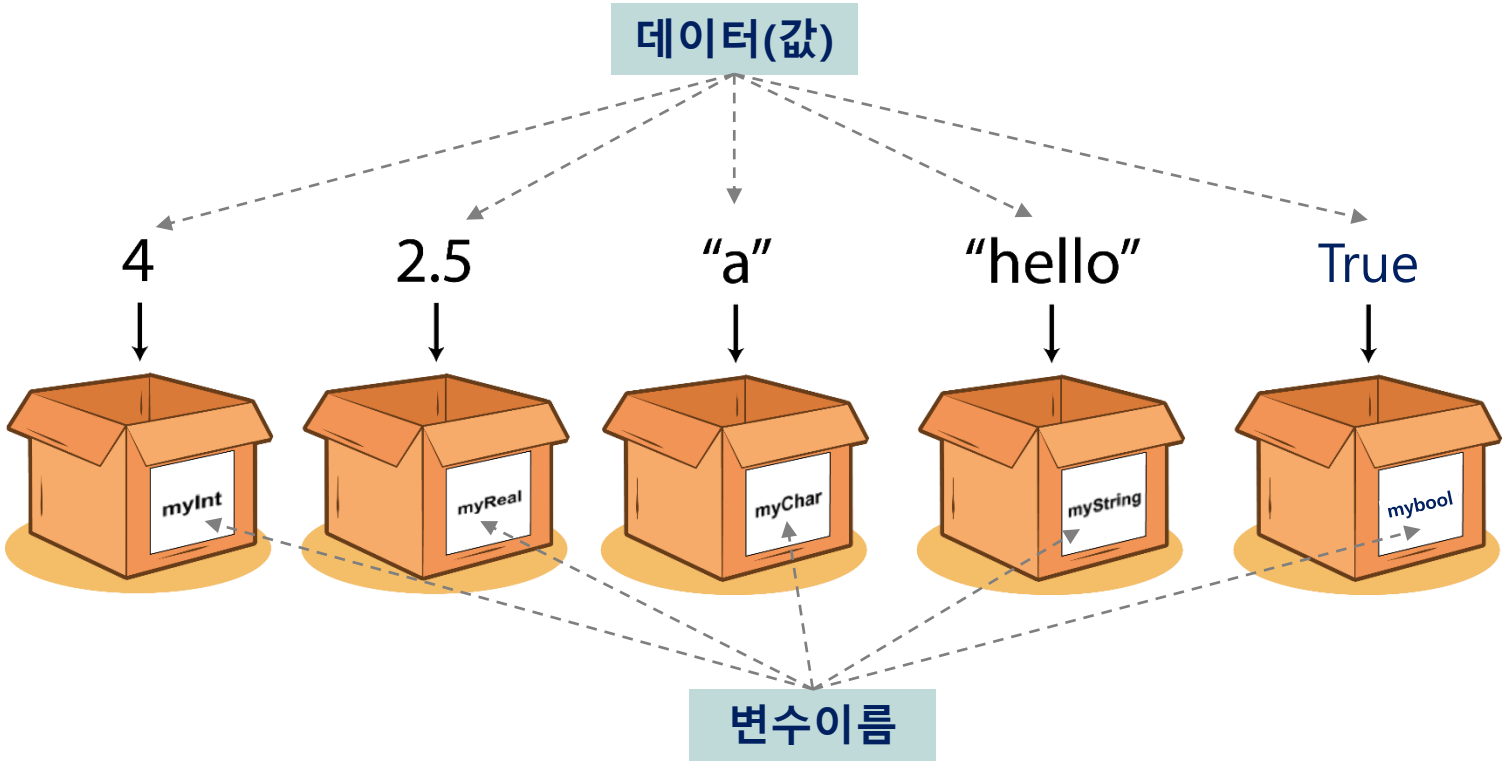
개발환경 설정



자료형과 변수

변수

✓ 데이터(값)를 저장하는 메모리 공간



변수

- ✓ 파이썬에서 변수는 값을 가리키는 이름이다.
- ✓ 파이썬에서는 변수에 값이 할당될 때 그 값의 데이터 형에 따라 변수 타입이 결정된다.
- ✓ 변수 이름은 영문, 숫자, _(underbar기호)로 만들 수 있고 숫자로 시작할 수 없다.
- ✓ 변수 이름은 영문의 대, 소문자를 구별한다.

자료형

- ✓ 프로그램에서 사용할 수 있는 자료의 종류
- ✓ 변수에는 정수, 실수, 문자열 등의 다양한 값들을 대입하여 저장할 수 있음

데이터 타입	의미	예시
정수(integer)	소수점이 없는 수	-5, 100, 12345
실수(floating-point)	소수점이 있는 수	-5.0, 3.14, 95.533
문자열(string)	따옴표로 에워싼 문자 또는 숫자	'Python', "123"
논리(boolean)	논리 값	True, False

기본 자료형

정수형

int

소수점이 없는 수

- 10진수
- 2진수 (예:0b1010)
- 8진수(예:0o71)
- 16진수(예:0xff)

실수형

float

소수점이 있는 수

- 지수 표현 : e, E사용
- $3.4*10^{10}$ -> 3.4e10
- $3.4*10^{-10}$ -> 3.4e-10

문자열형

str

하나 또는 다수의
문자로 구성된 형

문자열 앞 뒤에
쌍따옴표(")나
홑따옴표(')를 붙인다.

부울형

bool

True / False
두 가지 값을 가진다.

숫자형 데이터 타입

- ✓ 소수점이 없는 숫자 데이터 '정수형'과 소수점이 있는 숫자 데이터 '실수형'
- ✓ 변수에 저장할 때는 숫자만 입력함
- ✓ '실수형'은 소수점 16번째 자리까지만 메모리에 저장하고 나머지는 손실됨.
  (메모리의 크기에 제약을 받기 때문에)

항목	파이썬 사용 예
정수	100, -365, 0
실수	100.12, -3.14, 3.4e10
8진수	0o15, 0o36
16진수	0x1B, 0xFF

자료형 변환

✓ 자료형은 서로 간에 변환이 가능하며, '형 변환' 이라고도 함

✓ 자료형을 변환하기 위해서는 함수를 사용함

- 각 함수의 괄호 안에 데이터를 넣으면 데이터의 자료형이 변환됨

함수	의미	사용 예
str( )	문자형으로 변환	str(5), str(12.34), str(True)
int( )	정수형으로 변환	int('100'), int(True)
float( )	실수형으로 변환	float('3.14')
bool( )	논리형으로 변환	bool('True'), bool(0)

연산자

✓ 수식 : 연산자와 피연산자로 이루어지는 계산식

- 연산자 : 컴퓨터에 복잡한 계산을 시키기 위해 사용함
- 피연산자 : 연산의 대상, 연산에 필요한 데이터

[수식]

연산자의 종류

✓ 사칙연산을 비롯한 다양한 산술 연산자와 비교 연산자, 논리 연산자, 할당(대입) 연산자 등을 사용

구분	연산자	설명
산술 연산자	+, -, *, /, %, //, **	덧셈, 뺄셈 등 산술 연산을 한다.
할당 연산자	=, +=, -=, *=, /=, %=, //=, **=	오른쪽의 값을 연산하여 왼쪽에 대입한다.
비교 연산자	==, !=, >, >=, <, <=	크고 작음을 비교한다.
논리 연산자	and, or, not	참(True)과 거짓(False)을 판별한다.

산술 연산자

✓ 덧셈, 뺄셈, 곱셈, 나눗셈의 사칙연산 및 몫과 나머지 연산 등

연산자 기호	의미	예	결과
+	더하기	5 + 2	7
-	빼기	5 - 2	3
*	곱하기	5 * 2	10
/	나누기	5 / 2	2.5
%	나머지	5 % 2	1
//	몫	5 // 2	2
**	거듭제곱	5 ** 2	25

비교 연산자

✓ 두 개의 숫자를 비교하는 연산자. 연산 결과는 부울(bool)형

비교 연산자	의미	예	설명
==	같다	a == b	a는 b와 같다
!=	같지 않다	a != b	a는 b와 같지 않다
<	작다	a < b	a는 b보다 작다
>	크다	a > b	a는 b보다 크다
<=	작거나 같다	a <= b	a는 b보다 작거나 같다
>=	크거나 같다	a >= b	a는 b보다 크거나 같다

논리 연산자

✓ 어떤 조건을 만족하는 참(True)과 거짓(False)을 이용해 연산하는 것

- 논리 연산을 하기 위해서는 자료형이 부울(bool)형이어야 한다.

논리 연산자	의미	예	설명
and	논리곱	A and B	A와 B가 모두 참일 때만 참
or	논리합	A or B	A와 B가 모두 거짓일때만 거짓 (둘 중 하나라도 참이면 결과는 참)
not	논리 부정	not A	A가 참이면 거짓, 거짓이면 참

논리 연산자

논리 연산자를 이용한 진리표

A	B	A and B	A or B	not A
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

문자열 자료형

✓ 문자 형태로 이루어진 자료형

```
"Life is too short, You need Python"
"a"
"123"
```

항목	파이썬 사용 예
문자	“A”
문자열	“apple” ‘I love Python'

- 파이썬에서는 문자와 문자열을 구분하지 않는다.
- 문자(열) 자료는 따옴표 사이에 넣어 데이터임을 표시한다.

문자열 인덱싱(Indexing)

- ✓ 문자열 내의 어떤 문자 값을 가져오기 위해서 문자열의 모든 문자마다 위치 값을 주는 것
- ✓ 중괄호[ ]를 사용하고 인덱스는 0부터 시작한다.
- ✓ s[13]의 값은 h, (s[-5]의 값은?)

s라는 문자열	L	i	f	e		i	s		t	o	o		s	h	o	r	t	,
인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
음의 인덱스	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

뒤에서부터 인덱싱을 하려면 음수를 사용.
-1부터 시작

문자열 슬라이싱(Slicing)

- ✓ 문자열 내의 일부분의 값(하나 이상의 문자 값)을 가져오는 것
- ✓ [시작 인덱스 : 끝 인덱스 : (간격)] 에서 인덱스의 범위 지정 (시작 <= s < 끝)
- ✓ 처음 인덱스(0)와 끝 인덱스는 생략할 수 있다.
- ✓ 'Life'를 가져오려면 : s[0:4] 또는 s[:4]

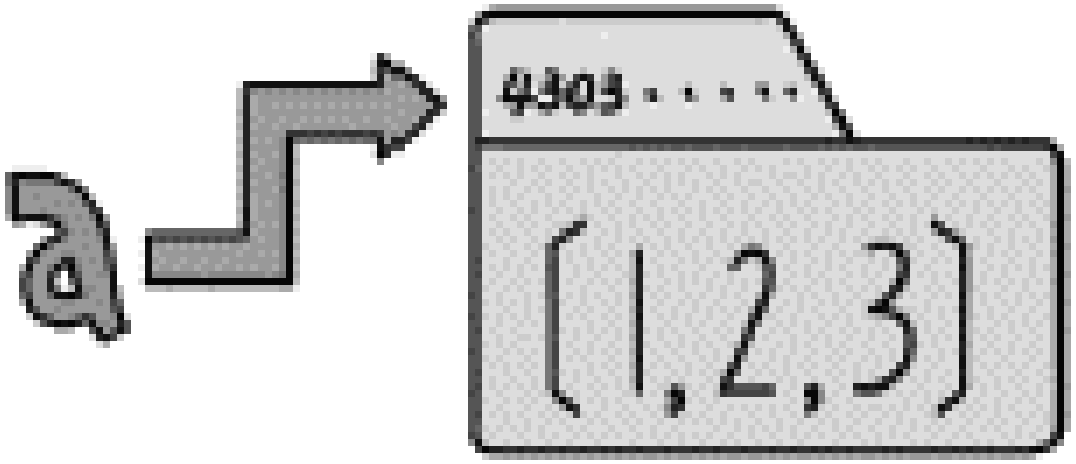
L	i	f	e		i	s		t	o	o		s	h	o	r	t	,
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

변수

✓ 변수?

객체를 가리키는 것

객체란 우리가 지금까지 보아 온 자료형의 데이터(값)와 같은 것을 의미



입력

✓ 사용자가 입력한 데이터는 문자열 형태로 반환(저장)됨.

✓ 따라서, 입력 받은 데이터를 이용하여 계산하려면 숫자형으로 변환 해야함.

- int( ) 함수 : 문자열을 **정수**로 변환

  사용 방법1) 변수 = int(input("프롬프트"))

  사용 방법2) 변수 = input("프롬프트")

  변수 = int(변수)

- float( ) 함수 : 문자열을 **실수**로 변환

  예) 변수 = float(input("프롬프트"))

```
1 # 키보드 입력
2
3 name = input('이름을 입력하세요: ')
4 print(name, '님 안녕하세요?')
```

```
이름을 입력하세요: 해리
해리 님 안녕하세요?
```

표준 출력함수 : print( ) 함수

✓ 실행한 결과를 기기(모니터나 스마트폰 등)의 화면을 통해 보여 주는 함수

print(값, …, [sep=‘ ‘], [end=‘ ‘])

- 값 : 출력 대상(숫자, 문자열, 변수, 계산식)
- sep : 출력 대상들 사이에 넣을 구분 기호 지정, 기본값은 공백
- end : 값 출력하고 마지막에 출력할 문자열 지정, 기본값은 줄바꿈

출력형태 - 콤마(,)로 구분하여 출력하는 형태

✓ **제어 문자, 이스케이프 시퀀스(escape sequence)**

- 화면에 출력되지는 않지만 출력 결과를 제어하는 문자
- \ (₩, 백슬래시) 뒤에 문자나 숫자가 오는 조합

이스케이프 시퀀스	표현 내용	설명
\n	새로운 줄	▪ 새로운 줄(New Line)을 의미함 ▪ Enter 키를 누른 것과 같은 줄바꿈 효과
\t	탭	▪ Tab 키를 누른 효과(보통 4칸 띄움)
\’	작은따옴표 문자	▪ ’ (작은 따옴표) 자체를 출력함
\”	큰따옴표 문자	▪ ” (큰 따옴표) 자체를 출력함
\\	\ 문자	▪ \ (백슬래시) 자체를 출력함

출력형태 - 서식 지정자를 이용하는 형태

✓ 서식 지정자 : %로 시작하고 자료형을 뜻하는 문자로 구성됨

- ' %자료형 ' % (값)

서식 지정자	설명	예시	실행 결과
%d	정수	'합계 = %d 점' % (289)	합계 = 289 점
%f	실수	'%f와 %5.1f' %(3.14,3.14)	3.140000와 3.1
%s	문자열	'Hello %s' %('Python')	Hello Python
%c	문자	'90 이상은 %c 등급' %('A')	90 이상은 A등급
%x	16진수	'100은 16진수로 %x' %(100)	100은 16진수로 64
%o	8진수	'100은 8진수로 %o' %(100)	100은 8진수로 144
%e	지수	'100은 %e' %(100)	100은 1.000000e+02

출력형태 - format( ) 함수를 이용하는 형태

✓ { }(중괄호)와 format( ) 함수 안의 값들을 순서대로 대응하여 출력하는 형태

- '{인덱스}' . format(값)

- '{인덱스 : 서식지정자}' . format(값)

```
1 # 두수를 입력받아 곱 계산
2 # format() 함수 이용, 계산결과 천단위 쉼표(,) 표시
3
4 a = int(input("첫 번째 정수 입력: "))
5 b = int(input("두 번째 정수 입력: "))
6
7 print('{0} * {1} = {2:,}'.format(a,b,a*b))
```

실행 결과

```
첫 번째 정수 입력: 100
두 번째 정수 입력: 234
100 * 234 = 23,400
```

출력형태 - f-문자열(f-string) 을 이용하는 형태

- ✓ format( ) 함수 업그레이드 버전, 파이썬 버전 v3.6 이상부터 사용 가능
- ✓ f를 접두사와 출력하고 싶은 '값'을 { }(중괄호)로 감싸서 넣는 방법

- f '{값:서식지정자}'

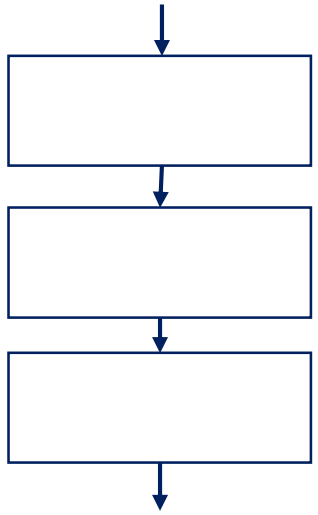
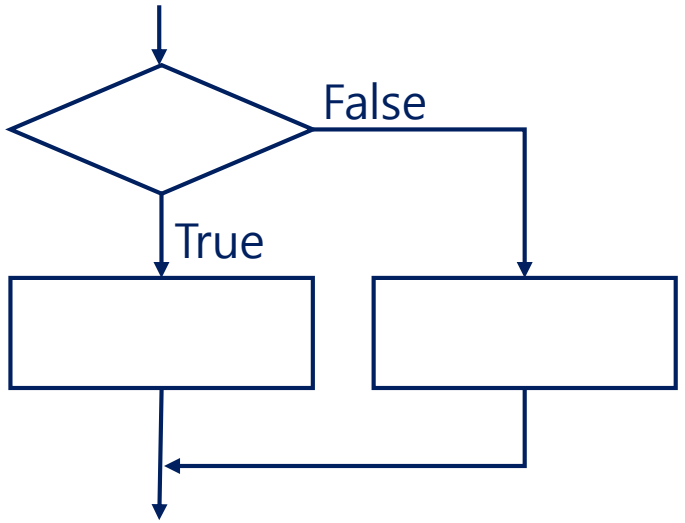
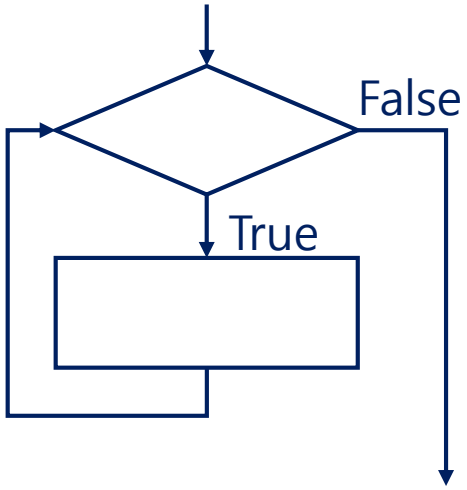
```
1 # 두수를 입력받아 곱 계산
2 # f-string 이용
3
4 a = float(input("첫 번째 값 입력: "))
5 b = float(input("두 번째 값 입력: "))
6
7 print(f'{a:.2f} * {b:.3f} = {a*b:,.2f}')
```

실행 결과

```
첫 번째 값 입력: 1000.1234
두 번째 값 입력: 2000.456789
1000.12 * 2000.457 = 2,000,703.65
```

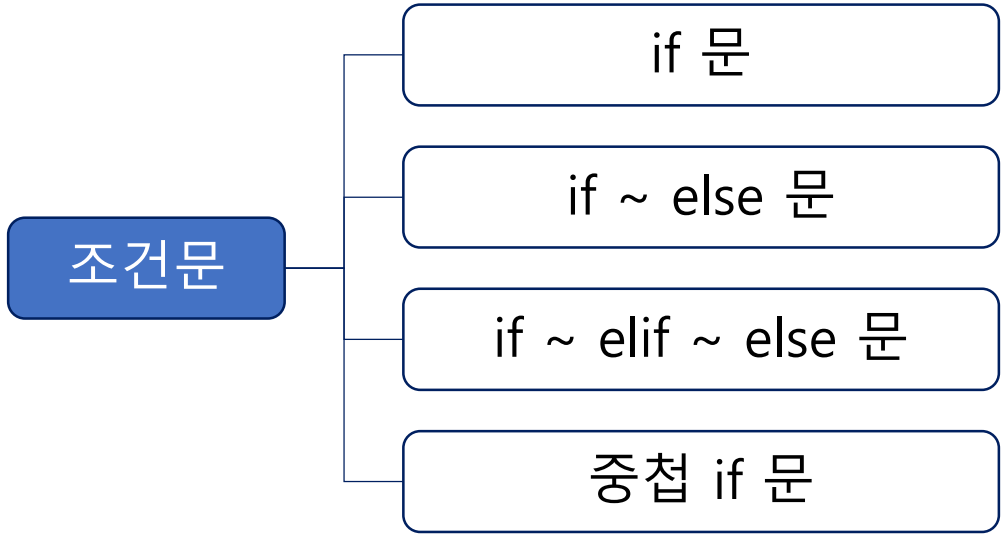
프로그램의 3가지 기본 제어 구조

✓ 제어구조 : 프로그램에서 해야 할 동작의 흐름이나 순서를 처리하는 방법

순차구조	선택구조	반복구조
명령들이 순차적으로 실행되는 구조	조건에 따라 명령을 선택하여 실행되는 구조	동일한 명령이 반복되면서 실행되는 구조
		

조건문

- ✓ 조건문은 특정 조건을 만족할 때 어떤 동작을 수행하도록 하는 문장
- ✓ 조건식의 참(True)과 거짓(False)의 상태에 따라 그 상태에서 필요로 하는 코드들을 수행



if 문의 조건식

✓ 관계 연산자나 논리 연산자 등이 사용된 수식

✓ 관계 연산자 : 두 개의 피연산자를 비교

연산	표현 내용	결과 (x = 10, y = 3)
x > y	x가 y보다 큰가?	True
x >= y	x가 y보다 크거나 같은가?	True
x < y	x가 y보다 작은가?	False
x <= y	x가 y보다 작거나 같은가?	False
x == y	x와 y가 같은가?	False
x != y	x와 y가 다른가?	True

if 문의 조건식

✓ 논리 연산자 : 여러 조건을 조합하여 참인지, 거짓인지 파악

논리 연산자	의미
x and y	x와 y가 모두 True이면 True, 그렇지 않으면 False 예) 국어 점수가 80점 이상이고 영어 점수가 80점 이상이면 : kor >=80 and eng >=80
x or y	x와 y중에서 하나만 True이면 True, 모두 False이면 False 예) 국어 점수가 80점 이상이거나 영어 점수가 80점 이상이면 : kor >=80 or eng >=80
not x	x와 True이면 False, x가 False이면 True

조건식 예시

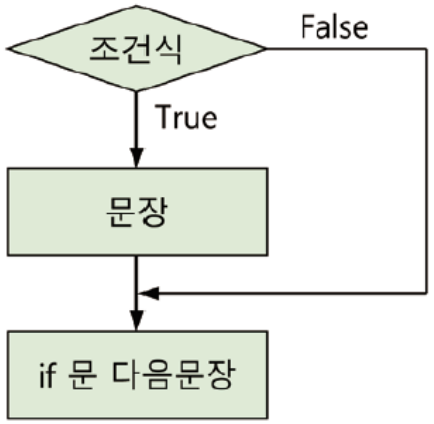
조건식	설명
total >= 300	total 변수에 저장된 값이 300이상이면 True 값을 갖는 조건식
kor + math < 100	kor 변수와 math 변수 값을 더한 값이 100 미만이면 True 값을 갖는 조건식
6 <= age < 60 → age >=6 and age < 60	age 변수에 저장된 값이 6이상이고 60 미만이면 True 값을 갖는 조건식
num % 3 == 0	num 변수에 저장된 값을 3으로 나눈 나머지가 0이면 True 값을 갖는 조건식
grade == 'a' or math >= 90	grade 변수에 저장된 값이 'a' 이거나, math 변수에 저장된 값이 90 이상이면 True 값을 갖는 조건식
not(weather == '여름')	weather 변수에 저장된 값이 '여름'이 아니면 True 값을 갖는 조건식

if 문

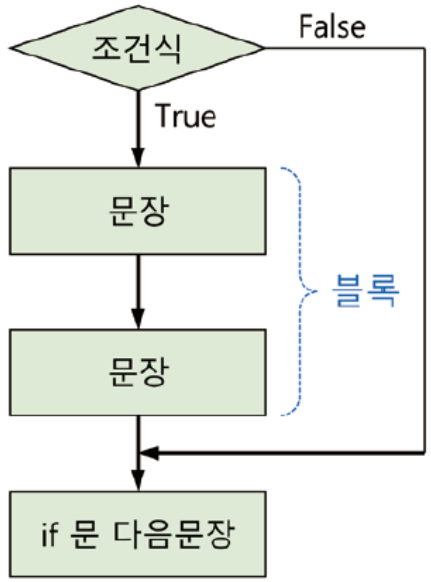
✓ 선택 구조를 위한 기본적인 문장

✓ 조건에 맞으면(조건식의 값이 참(True)이면) 문장/블록을 실행하고, 그렇지 않으면 건너뜀

```
if 조건식:
    문장
```



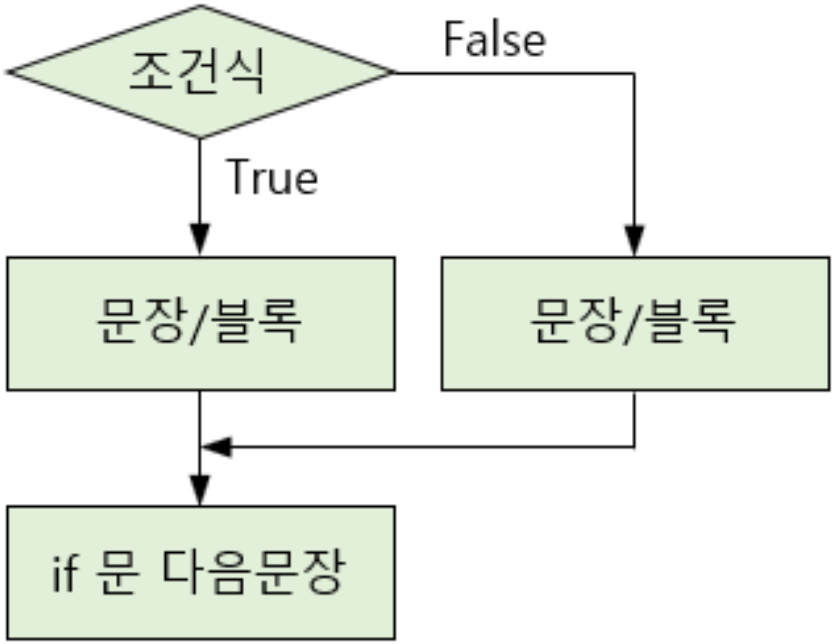

```
if 조건식:
    블록
```



if ~ else 문

- ✓ 조건식의 값이 참과 거짓일 경우 구분하여 실행
- ✓ 참이나 거짓에 해당하는 부분을 반드시 한 부분은 실행

```
if 조건식:
    문장(또는 블록)
else:
    문장(또는 블록)
```

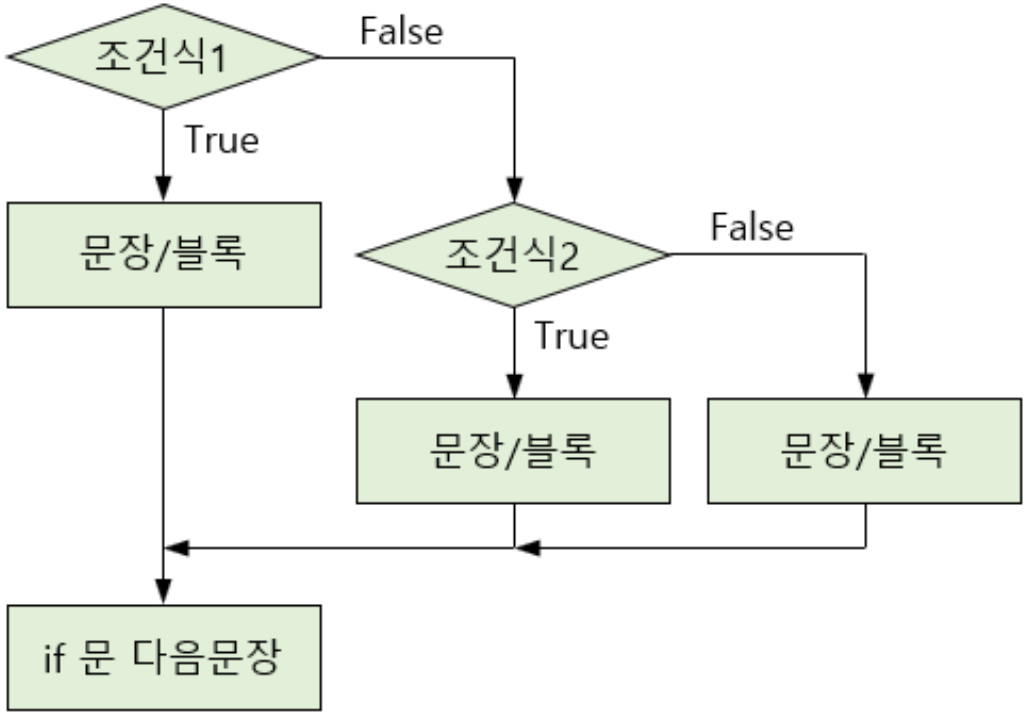


if~elif 문

✓ 다중 비교를 할 때 사용하는 조건문

✓ elif는 else if의 줄임말로 '그렇지 않고 ~ 라면'

```
if 조건식1:
    문장(또는 블록)
elif 조건식2:
    문장(또는 블록)
else:
    문장(또는 블록)
```

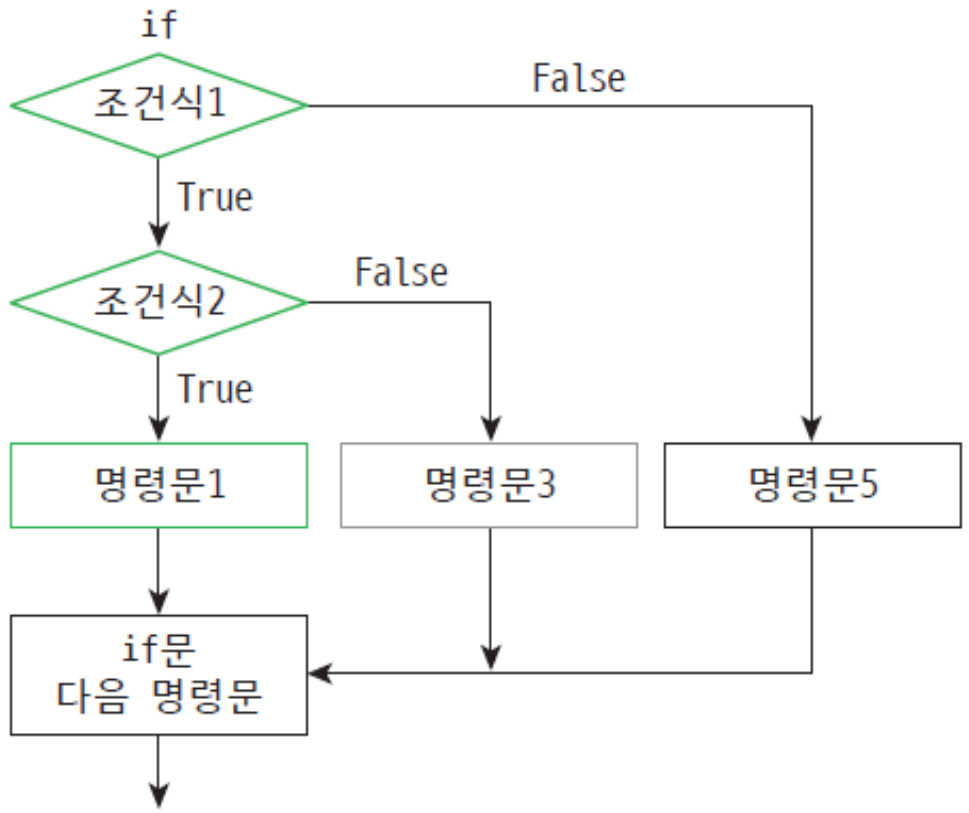


중첩 조건문(중첩 if 문)

✓ 조건문 내에 또 다른 조건문을 사용한 조건문

[if 문의 조건식이 참인 경우 또 다른 if 문이 연속하여 있는 경우]

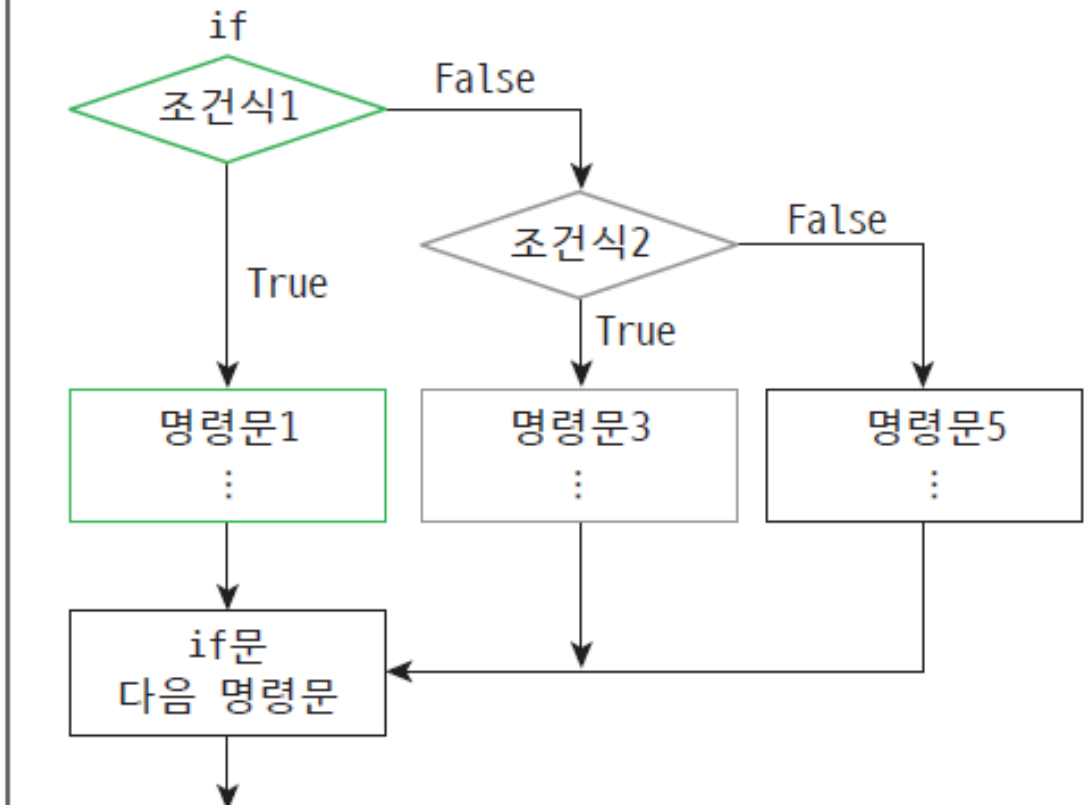
```
if 조건식1:
(공백) if 조건식2:
(공백) (공백) 명령문1
          ⋮
(공백) else:
(공백) (공백) 명령문3
          ⋮
else:
(공백) 명령문5
          ⋮
if문 다음 명령문
```



중첩 if 문

✓ [if 문의 조건식이 거짓인 경우 else 문 안에 또 다른 if 문이 있는 경우]

```
if 조건식1:
(공백) 명령문1
        ⋮
else:
(공백) if 조건식2:
(공백) (공백) 명령문3
        ⋮
(공백) else:
(공백) (공백) 명령문5
        ⋮
if문 다음 명령문
```



정리하기

- if 문

```
if 조건식:
    문장(블록)
```

- if ~ else 문

```
if 조건식:
    문장(또는 블록)
else:
    문장(또는 블록)
```

- if ~ elif ~ else 문

```
if 조건식1:
    문장(또는 블록)
elif 조건식2:
    문장(또는 블록)
else:
    문장(또는 블록)
```

- 조건식

- 관계연산자
  : >, <, >=, <=, ==, !=

- 논리연산자 : and, or, not

- 중첩 if 문

```
if 조건식1:
    if 조건식2:
        문장(블록)
    ⋮
    else:
        문장(블록)
    ⋮
else:
    문장(블록)
    ⋮
```

반복문의 필요성

- ✓ 반복(iteration)은 동일한 문장을 여러 번 반복시키는 구조
- ✓ 반복문은 컴퓨터가 같은 일을 반복할 수 있도록 하는 명령문, 루프문(loop statement)이라고도 함
- ✓ 반복적인 업무를 쉽고 빠르게 처리하기 위해 사용함
- ✓ 예를 들어, '안녕' 5번 반복해야한다면?

```
print("안녕!")
print("안녕!")
print("안녕!")
print("안녕!")
print("안녕!")
```

'안녕' 1000번 반복해야한다면? →

```
print("안녕!")
print("안녕!")
print("안녕!")
…
…
print("안녕!")
```

1000번 출력

[비효율적 코드]

반복문 사용 →

```
for i in range(1000):
    print( " 안녕!")
```

반복문 종류

	조건 반복	횟수 반복	무한 반복
반복 기준	조건의 충족	반복 횟수	무한 실행
사용 예	지칠 때까지 반복하라	10번 반복하라	계속하라
반복문	**while**	**for**	**while True**

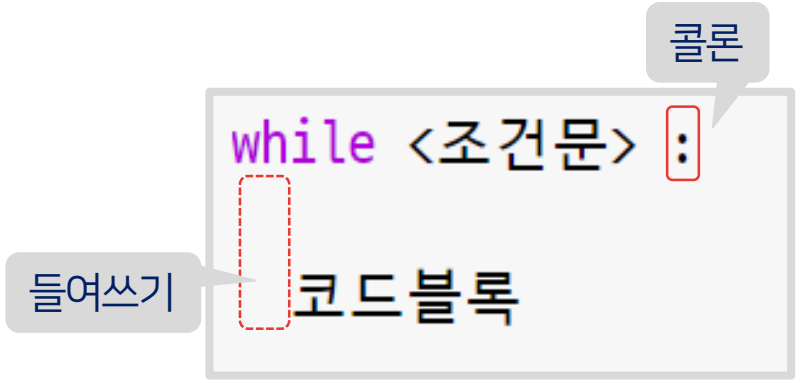
while문 : 조건식 결과에 따른 반복

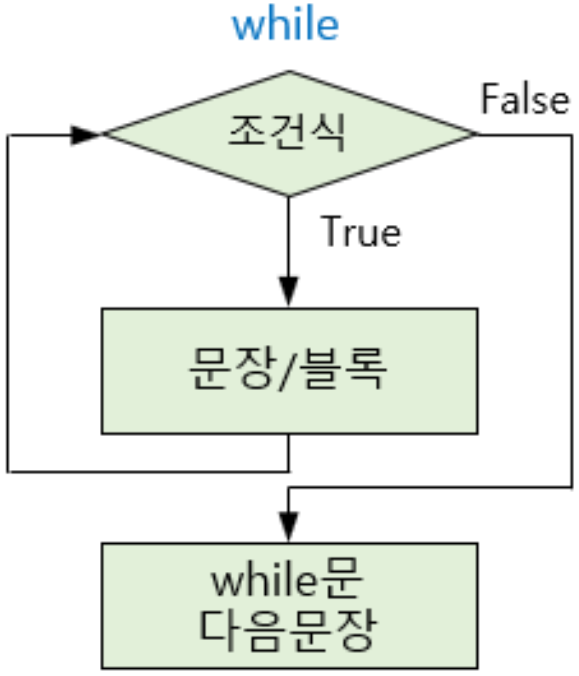
for문 : 횟수에 따른 반복

while 문

- ✓ **조건**에 따른 반복
- ✓ 조건식의 값이 참(True)인 경우 문장(또는 블록)을 반복하고,
  조건식의 값이 거짓(False)이면 반복을 종료하여 빠져나감

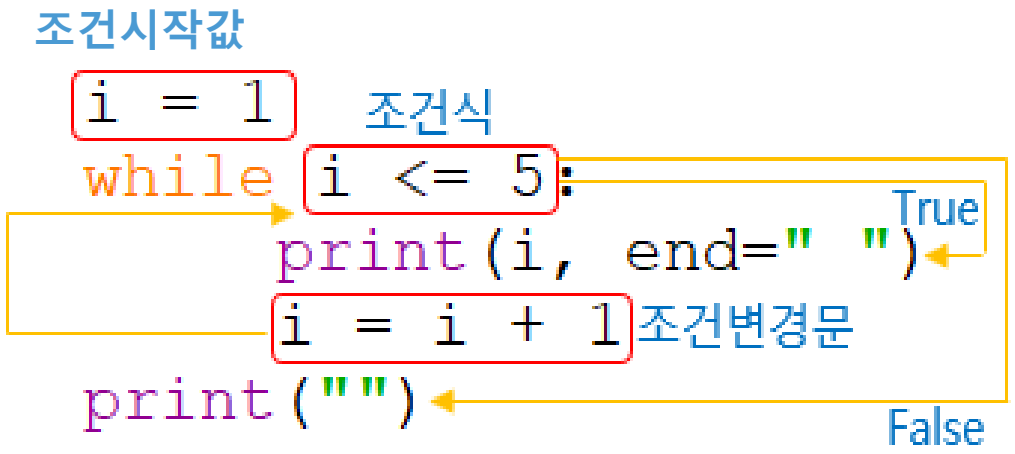
```
while 조건식:
    문장(또는 블록)
```





while 문

예) 1부터 99까지 홀수값 출력




```
조건시작값
while 조건식 :
    문장(또는 블록)
    조건변경문
```

for 문

✓ 지정된 횟수만큼 반복하는 횟수 제어 반복

```
for 변수 in 이터러블 :
    반복할 문장
```

- 이터러블(iterable) : 반복 가능한 객체

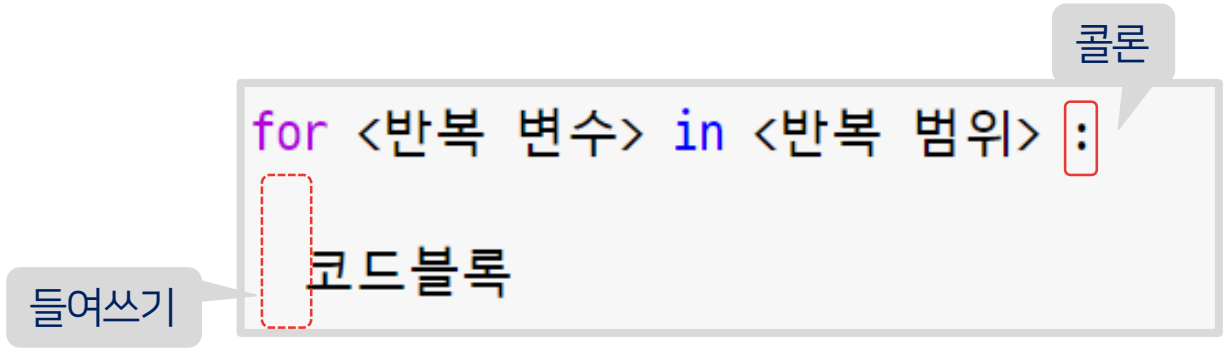
✓ for 문의 2 가지 형식

① for 변수 in 시퀀스(문자열 또는 리스트)

② for 변수 in range(정수)

for 문

- ✓ <반복 범위>는 1개 이상의 데이터이고 반복 변수에 대입할 수 있다
- ✓ <반복 변수>는 <반복 범위>에 따라 변하면서 <코드 블록>을 실행한다.
- ✓ <코드 블록>에서는 <반복 변수>를 이용할 수 있다.



range( ) 함수 예시

- ‘시작값’에서 시작하여 ‘종료값-증가값’ 까지 ‘증가값’ 간격으로 정수들 생성
- ‘시작값’과 ‘증가값’ 을 생략하면 ‘시작값'은 0, ‘증가값'은 1로 간주됨

range( ) 함수	생성 정수들	설명
range(0, 8, 2)	[0, 2, 4, 6]	▪ 0부터 시작하여 ‘종료값(8) – 증가값(2)’ 까지 2씩 증가하는 정수들
range(2, 8)	[2, 3, 4, 5, 6, 7]	▪ 2부터 시작하여 ‘종료값(8) – 증가값(1)’ 까지 1씩 증가하는 정수들
range(6, 0, -1)	[6, 5, 4, 3, 2, 1]	▪ 6부터 시작하여 ‘종료값(0) – 증가값(-1)’ 까지 1씩 감소하는 정수들
range(5)	[0, 1, 2, 3, 4]	▪ range(0, 5, 1) 과 같은 의미 ▪ 0부터 시작하여 ‘종료값(5) – 증가값(1)’ 까지 1씩 증가하는 정수들

range([시작 값], [끝 값], [증가 값step])

for 문 예시

✓ range( ) 함수를 이용하여 0부터 4까지 출력

```
for i in range(5):
    print(i, end=" ")
```

실행 결과

```
0 1 2 3 4
```

✓ range( ) 함수를 이용하여 1부터 10까지 합 출력

```
1 hap = 0
2 for i in range(1,11):
3   hap = hap +i
4 print(f"1부터 10까지 합 = {hap}")
```

break 키워드

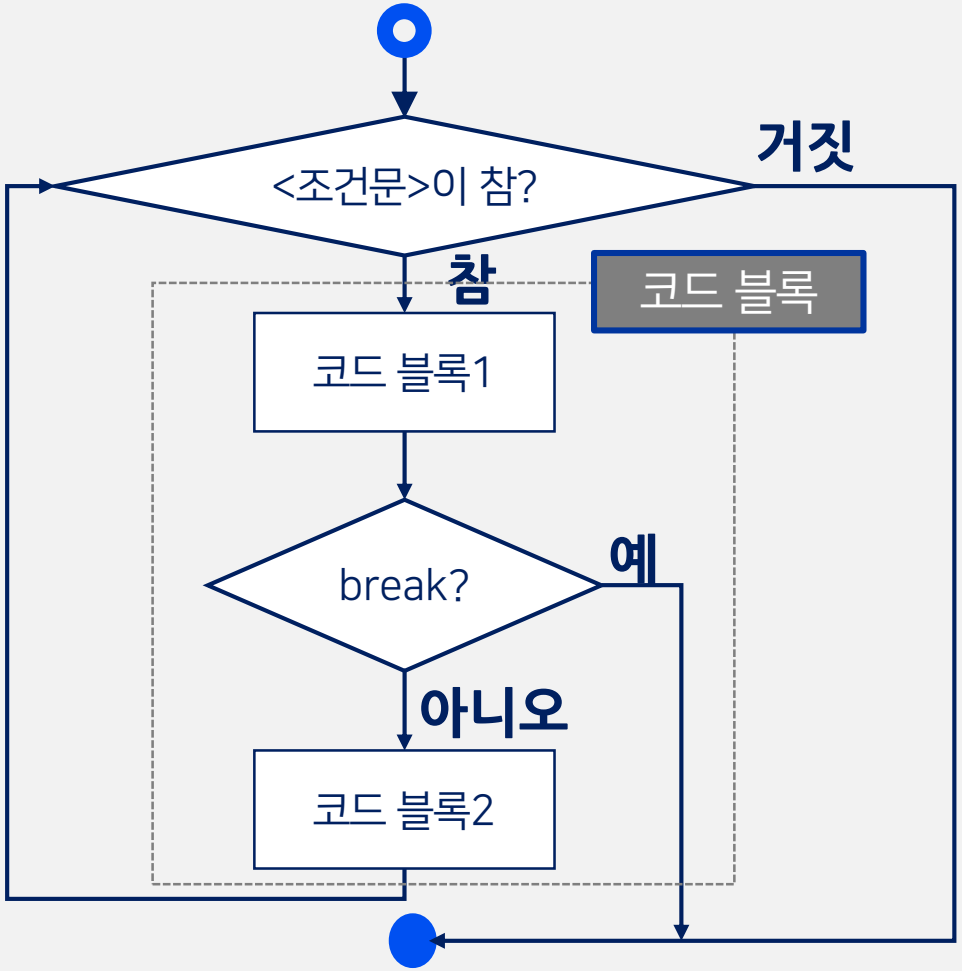
- 실행을 중단하고 반복문을 빠져나옴

  예) 1~10까지 더하되, 합이 20 이상이 될 때 정수값 찾기

```
1 # 1~10까지 더하되, 합이 20 이상이 될 때 정수값 찾기
2
3 hap =0
4 for i in range(1,11):
5   hap = hap + i
6   if hap >= 20:
7     print(f"정수:{i}, 합:{hap}")
8     break
```

실행 결과

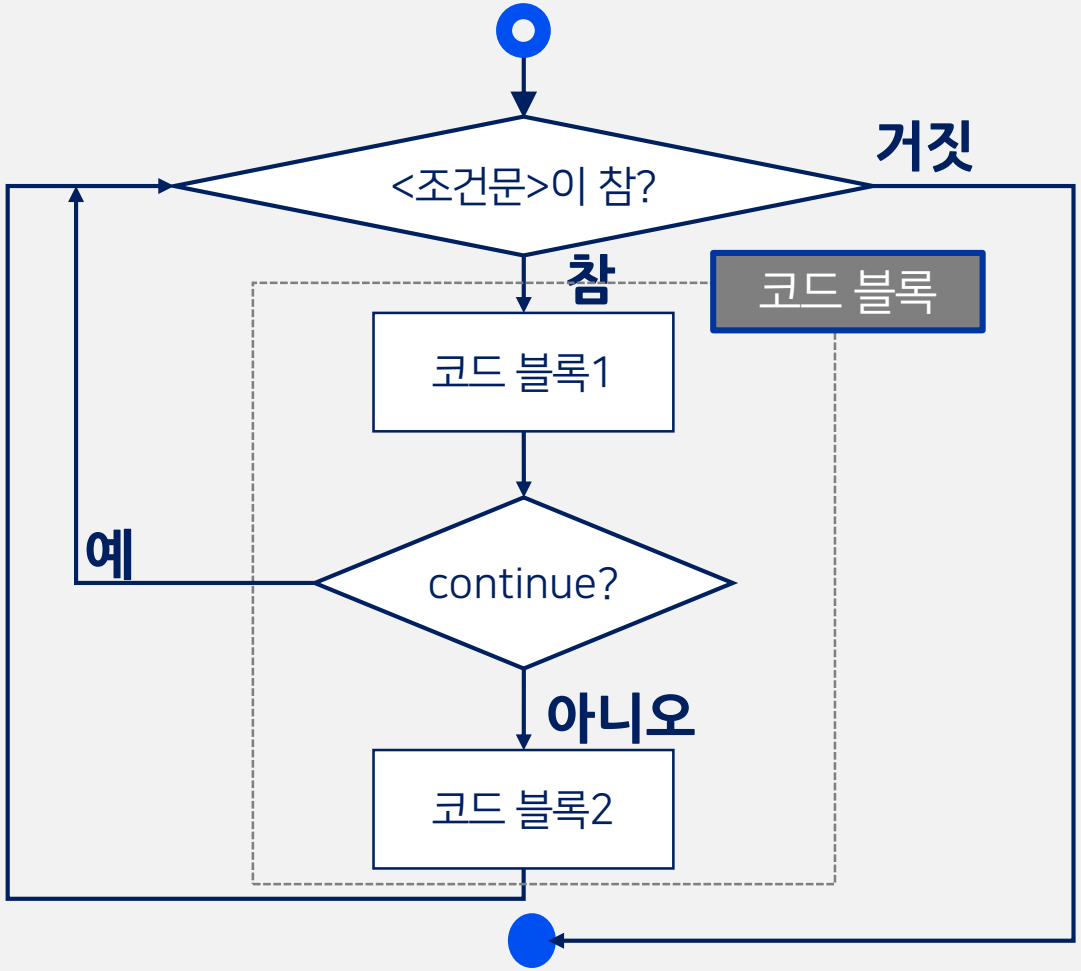
```
정수:6, 합:21
```



continue 키워드

예) 1~10까지의 정수 중 홀수와 홀수 합 출력

```
1 # 1~10까지의 정수 중 홀수와 홀수 합 출력
2 # continue 키워드
3
4 hap =0
5 for i in range(1,11):
6   if i % 2 == 0:
7     continue
8   print(i, end=" ")
9   hap = hap + i
10
11 print(f"₩n1~10까지의 정수 중 홀수 합 : {hap}")
```



실행 결과

```
1 3 5 7 9
1~10까지의 정수 중 홀수 합 : 25
```

복합 자료형

자료형

✓ 자료형 : 프로그래밍을 할 때 쓰이는 숫자, 문자열 등 자료 형태로 사용하는 모든 것

[변수 사용]

```
# 첫 번째 연락처
name_1  = '박연오'
phone_1 = '01012345678'

# 두 번째 연락처
name_2  = '이진수'
phone_2 = '01011001010'

# 세 번째 연락처 (비어 있는 자리)
name_3  = None
phone_3 = None

... (계속)
```

[문제점]

① 데이터가 추가되면 추가하려는 만큼 코드를 작성해야 한다.

② name와 phone은 연락처라는 한 쌍을 이루는 데이터 구성요소인데, 각각 다른 변수로 별개의 데이터로 저장된다.

[출처] https://velog.io/@suasue/Python-%EC%BB%AC%EB%A0%89%EC%85%98-%EC%9E%90%EB%A3%8C%ED%98%95-ktn11rmj

컬렉션 자료형

✓ 컬렉션 자료형 : 여러 개의 요소를 하나로 묶어 사용하는 자료형

컬렉션 자료형 종류	생성 방법	예시
리스트(list)	[ ]	address = [['박연호', '01012345678'], ['이진수, '01011000101']]
튜플(tuple)	( )	name = ('박연호', '이진수', '유재석')
딕셔너리(dictionary)	{키 : 값}	address = { '박연호' : '01012345678', '이진수' : '0101100101' }
세트(set)	{ }	phone = { '01012345678', '01011000101', '0107801234' }

리스트(list) 생성

✓ 여러 개의 데이터를 하나의 이름으로 저장하는 형태

```
리스트명 = [값1, 값2, ... ]
```

✓ 리스트에 저장되는 각각의 데이터를 항목 또는 원소라고 하며,
항목은 숫자나 문자, 다른 리스트 등 다양한 종류로 구성

예) score = [90, 85, 79, 93, 87, 98, 60]
data = ['홍길동', '김유신', 3.5, 6, ['a', 'b']]

✓ 각 항목은 순서대로 인덱스라고 하는 번호가 정해지며 0부터 시작

리스트 다루기

✓ 리스트 인덱싱(Indexing)

```
리스트명[인덱스]
```

✓ 리스트 슬라이싱(Slicing)

- 슬라이싱은 리스트 안에서 범위를 지정하여 원하는 요소들을 선택하는 연산

리스트 슬라이싱	설명
리스트명[인덱스1 : 인덱스2]	인덱스1 요소부터 인덱스2 앞 요소까지 선택
리스트명[인덱스1 :]	인덱스1부터 마지막까지 선택
리스트명[: 인덱스2]	첫 번째부터 인덱스2 앞 요소까지 선택
리스트명[:]	리스트 전체 요소 선택

✓ 리스트 값 변경하기

```
리스트명[인덱스] = 값
```

리스트 조작함수

✓ 리스트 변수 이름 뒤에 마침표(.)를 붙인 다음 함수 이름 사용

```
리스트명.함수명(인수)
```

1) 리스트 값 추가 : append( ) 함수, insert( ) 함수

```
1 fruit = ['사과', '바나나']
2
3  # append(값), 리스트 끝에 추가
4 print("== append(값)")
5 fruit.append('체리')
6 fruit.append(['배','딸기'])
7 print(fruit,end='₩n₩n')
8
9  # insert(인덱스, 값), 해당 위치에 추가
10 print("== insert(인덱스, 값)")
11 fruit.insert(1,'망고')
12 fruit.insert(4,'수박')
13 print(fruit)
```

실행 결과

```
== append(값)
['사과', '바나나', '체리', ['배', '딸기']]

== insert(인덱스, 값)
['사과', '망고', '바나나', '체리', '수박', ['배', '딸기']]
```

리스트 조작함수

2) 리스트 값 삭제 : pop( ) 함수, remove( ) 함수

```
1 fruit = ['사과', '망고', '바나나', '체리', '수박']
2
3  # pop() : 마지막 요소 삭제, pop(인덱스) : 해당 인덱스 요소 삭제
4 print("== pop(), pop(인덱스)")
5 fruit.pop()  # '수박' 요소 삭제
6 fruit.pop(1) # 1 인덱스 요소인 '망고' 삭제
7 print(fruit)
8
9  # remove(요소) : 해당 요소 삭제
10 print("₩n== remove(요소)")
11 fruit.remove('바나나')
12 print(fruit)
```

실행 결과

```
== pop(), pop(인덱스)
['사과', '바나나', '체리']

== remove(요소)
['사과', '체리']
```

리스트 조작함수

3) **리스트 정렬** : sort( ), sort(reverse=True) 함수, **리스트에서 찾을 값의 개수** : count(요소) 함수

```
1 score = [90,20,50,20,80,90,20]
2
3 # 리스트 정렬 : sort( ), sort(reverse=True) 함수
4 score.sort()
5 print("오름차순: ", score)
6 score.sort(reverse=True)
7 print("내림차순: ", score)
8
9 #리스트에서 찾을 값의 개수 : count(요소) 함수
10 print("요소 20 의 개수: ",score.count(20))
11
12 #리스트 요소 개수 : len(리스트) 함수
13 print("리스트 전체 개수: ", len(score))
```

실행 결과

```
오름차순:  [20, 20, 20, 50, 80, 90, 90]
내림차순:  [90, 90, 80, 50, 20, 20, 20]
요소 20 의 개수:  3
리스트 전체 개수:  7
```

리스트 조작함수

함수	설명	사용법
append()	리스트에 요소를 마지막 위치에 새로 추가	리스트.append(값)
insert()	리스트의 해당 위치에 요소를 새로 삽입	리스트.insert(위치,값)
sort()	오름차순정렬 내림차순정렬	리스트.sort() 리스트.sort(reverse=True)
reverse()	현재의 리스트를 그대로 거꾸로 뒤집는다.	리스트.reverse()
pop()	리스트 제일 뒤의 항목을 빼내고, 빼낸 항목은 삭제 제거할 위치에 있는 요소를 제거	리스트.pop() 리스트.pop(위치)
remove()	해당 요소를 찾아 삭제	리스트.remove(삭제할값)
count()	해당 요소의 개수를 반환	리스트.count(찾을값)
index()	리스트에 위치 값이 있으면 위치값을 반환	리스트.index(값)

리스트 적용 내장 함수

연산자	의미
len(리스트)	리스트에 포함된 원소들의 개수를 반환. 반복문을 사용하여 리스트 원소들을 하나씩 접근할 때 유용
max(리스트)	리스트에 포함된 원소들의 최대값을 반환
min(리스트)	리스트에 포함된 원소들의 최소값을 반환
sum(리스트)	리스트에 포함된 원소들의 합을 반환

리스트 적용 내장 함수

```
1 score = [90,20,50,20,80,90,20]
2
3 print("합 = ",sum(score))
4 print("개수 = ",len(score))
5 print(f"평균 = {sum(score)/len(score):.2f}")
6 print("최대값 = ",max(score))
7 print("최소값 = ",min(score))
```

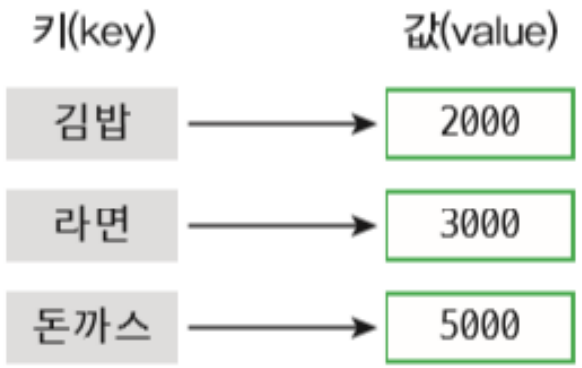
실행 결과

```
합 =  370
개수 =  7
평균 = 52.86
최대값 =  90
최소값 =  20
```

딕셔너리(dictionary)

✓ 순서가 없는 컬렉션 자료형으로 각각의 요소는 {키 : 값} 형태로 저장됨.

```
딕셔너리명 = {키1:값1, 키2:값2, 키3:값3, …}
```



menu = {'김밥':2000, '라면':3000, '돈까스':5000}

✓ 딕셔너리에서 **키는 중복할 수 없지만,** 값은 중복이 가능함

menu = {'김밥':3200, '라면':3000, '돈까스':5000, '야채김밥':3200}

딕셔너리(dictionary) 탐색

✓ 딕셔너리 조회

```
딕셔너리명 [키]
```

예) menu['라면']

✓ 딕셔너리 값 삽입 및 변경

- 새로 추가하는 키는 기존에 있던 키와 중복되어서는 안 됨
- 만약 키가 중복되면 기존 키에 저장되어 있는 값이 변경됨.

```
딕셔너리명 [키] = 값
```

예) menu['라면'] = 3500

딕셔너리(dictionary) 탐색

```python
menu = {'김밥':2000, '라면':3000, '돈까스':5000}

# 값 변경
menu['김밥'] = 3500

# 조회
print(f'김밥 값 : {menu["김밥"]}')

# 값 삽입
menu['야채김밥'] = 3500

print(menu)
```

실행 결과

```
김밥 값 : 3500
{'김밥': 3500, '라면': 3000, '돈까스': 5000, '야채김밥': 3500}
```

딕셔너리 조작 함수

✓ **딕셔너리 키(key), 값(value) 반환** : keys() 함수, values() 함수, items( ) 함수

- 딕셔너리.items() 함수 : 키(key)와 값(value)의 쌍을 튜플로 묶은 값을 반환

```
>>> adic = {'a': 90, 'b': 80, 'c': 60, 'd': 70}
```

```
>>> for x in adic.keys():
        print(x)
a
b
c
d
```

키 리스트

```
>>> for x in adic.values():
        print(x)
90
80
60
70
```

값 리스트

```
>>> for x in adic.items():
        print(x)
'a', 90
'b', 80
'c', 60
'd', 70
```

항목 리스트

딕셔너리 조작 함수

딕셔너리 키와 값 전체 출력

```
1 menu = {'김밥':2000, '라면':3000, '돈까스':5000}
2
3 # 방법1
4 print("==방법1")
5 for i in menu:    # 변수 i 에 menu 딕셔너리의 키값들 저장
6   print(f"{i} → {menu[i]}",end=" ")
7
8 # 방법2 : keys() 함수이용
9 print("==방법2 : keys()")
10 for i in menu.keys():    # 변수 i 에 menu 딕셔너리의 키값들 저장
11   print(f"{i} → {menu[i]}",end=" ")
12
13 # 방법3 : items() 함수이용
14 print("==방법3 : items() 함수이용")
15 for key,value in menu.items():
16   print(f"{key} → {value}",end=" ")
```

실행 결과

김밥 → 2000 라면 → 3000 돈까스 → 5000

딕셔너리 조작 함수

✓ **딕셔너리 요소 삭제 :** del(딕셔너리[키]) 함수, pop(키) 함수

```
1 menu = {'김밥': 3500, '라면': 3000, '돈까스': 5000, '야채김밥': 3500}
2
3 del(menu['야채김밥'])
4 menu.pop('돈까스')
5
6 print(menu)
```

실행 결과

```
{'김밥': 3500, '라면': 3000}
```

딕셔너리 메소드

함수	설명	사용법
get()	항목접근하기	딕셔너리.get(key)
pop() del ()	항목 꺼내고 삭제하기 항목삭제하기	딕셔너리.pop(key) del(딕셔너리[key])
items()	딕셔너리에 저장된 모든 항목	딕셔너리.items()
keys()	딕셔너리에 저장된 키	딕셔너리.keys()
values()	딕셔너리에 저장된 값	딕셔너리.values()