

파이썬 자료구조와 알고리즘 for Beginner

[강의교안 이용 안내]

- 본 강의교안의 저작권은 한빛아카데미(주)에 있습니다.
- 이 자료는 강의 보조자료로 제공되는 것으로 무단으로 전제하거나 배포하는 것을 금합니다.

02

CHAPTER

파이썬 기초 문법과 데이터 형식

학습목표

- 파이썬의 변수를 이해한다.
- 파이썬 연산자, 조건문, 반복문, 함수 등 문법을 익힌다.
- 자료구조를 위한 파이썬 기본 데이터 형식을 이해한다.
- 리스트, 딕셔너리, 튜플 등 연속형 데이터 형식을 이해한다.

SECTION 00 파이썬 이름 이야기

SECTION 01 파이썬의 기초 문법

SECTION 02 파이썬의 데이터형

연습문제

응용예제



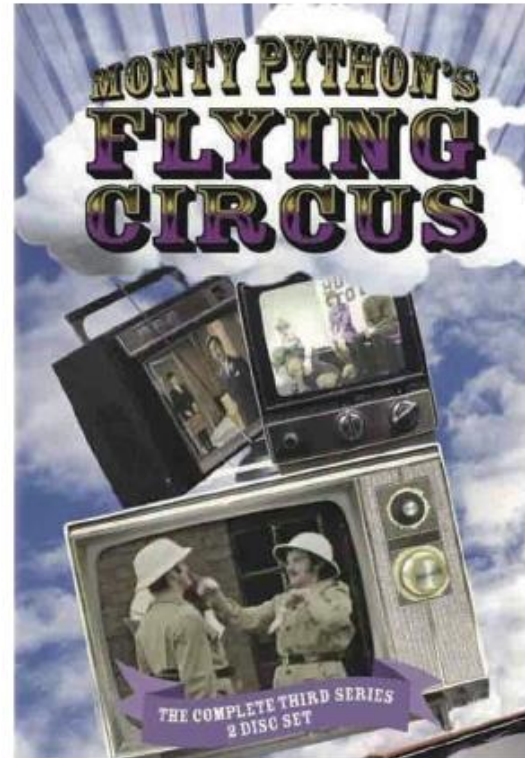
Section 00 파이썬 이름 이야기

■ 파이썬 이름의 유래

- 귀도 반 로섬 프로그래머가 개발한 프로그래밍 언어로 TV 코미디 프로그램 이름에서 따옴
- 파이썬(Python)은 비단뱀을 의미



파이썬 로고



파이썬 이름을 따온 [몬티 파이톤의 날아다니는 서커스] 프로그램

Section 01 파이썬의 기초 문법

■ 변수와 print() 함수

[변수와 연산자 동영상 강좌](#)

■ 변수 선언

- 한 행이나 여러 행에 나누어 변수에 값 대입 = 변수 선언 효과

```
boolVar, intVar, floatVar, strVar = True, 0, 0.0, ""
```

또는

```
boolVar = True  
intVar = 0  
floatVar = 0.0  
strVar = ""
```

- type() 함수를 사용해 bool(불), int(정수), float(실수), str(문자열)형 변수 생성 예

```
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

실행 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

Section 01 파이썬의 기초 문법

- print() 함수
 - 올바른 예

```
print("안녕하세요?")
```

“ ” 안의 내용을 화면에 출력

❶ `print("100")`

❶의 결과로 나온 100은 문자 100(일영영)

❷ `print("%d" % 100)`

❷의 결과로 나온 100은 숫자 100(백)

❸ `print("100+100")`

❸은 100+100을 출력

❹ `print("%d" % (100+100))`

❹은 숫자 100과 숫자 100을 더한 결과인 200을 출력


- 잘못된 예

❺ `print("%d" % (100, 200))`

❻ `print("%d %d" % (100))`

서식 개수와 따옴표 뒤에 나오는 숫자(또는 문자) 개수가 달라 오류 발생

`print("%d %d" % (100, 200))`



❺는 숫자 2개를 출력하려면 %d가 2개 필요
중간의 %는 왼쪽 서식과 오른쪽 값을 구분하는 역할
숫자가 여러 개이면 콤마(,)로 구분하여 ()로 감싸야 함

`print("%d" % (100))`

❻은 단순히 %d를 하나 삭제해야 올바른 표현

Section 01 파이썬의 기초 문법

- print() 함수
 - Print() 함수에서 사용할 수 있는 서식(표 2-1)

서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14, 3.14e3	실수(소수점이 붙은 수)
%c	'b', '한', '파'	문자 하나로 된 글자
%s	"안녕", 'abcdefg', "a"	한 글자 이상의 문자열

Section 01 파이썬의 기초 문법

- input() 함수
 - 키보드로 값을 입력 받음
 - 다음 예처럼 모든 것을 문자열로 간주하여 출력

```
var1 = input()  # Enter 후 100을 입력  
var2 = input()  # Enter 후 200을 입력  
  
print(var1 + var2)
```

실행 결과

100200

- 숫자로 입력 받으려면 다음 예처럼 int() 함수 사용

```
var1 = int(input())  # Enter 후 100을 입력  
var2 = int(input())  # Enter 후 200을 입력  
  
print(var1 + var2)
```

실행 결과

300

Section 01 파이썬의 기초 문법

- input() 함수
 - input("설명") 형태로 사용하면 입력 설명을 출력하여 사용자 입장에서 더 편리

```
var1 = int(input("숫자를 입력하세요 -->"))
```

실행 결과

숫자를 입력하세요 -->

Section 01 파이썬의 기초 문법

■ 연산자

■ 산술 연산자 종류(표 2-2)

연산자	설명	사용 예	
=	대입 연산자	a = 3	정수 3을 a에 대입
+	더하기	a = 5+3	5와 3을 더한 값을 a에 대입
-	빼기	a = 5-3	5와 3을 뺀 값을 a에 대입
*	곱하기	a = 5*3	5와 3을 곱한 값을 a에 대입
/	나누기	a = 5/3	5를 3으로 나눈 값을 a에 대입
//	나누기(몫)	a = 5//3	5를 3으로 나눈 후 소수점을 버린 값을 a에 대입
%	나머지 값	a = 5%3	5를 3으로 나눈 후 나머지 값을 a에 대입
**	제곱	a = 5**3	5의 3제곱을 a에 대입

■ 산술 연산자 사용 예

```
a = 5; b = 3  
print(a + b, a - b, a * b, a / b, a // b, a % b, a ** b)
```

a//b는 a를 b로 나눈 몫이고,
a%b는 a를 b로 나눈 나머지 값

실행 결과

```
8 2 15 1.6666666666666667 1 2 125
```

Section 01 파이썬의 기초 문법

- 대입 연산자 종류(표 2-3)

연산자	사용 예	
<code>+=</code>	<code>a += 3</code>	<code>a = a + 3</code> 과 동일
<code>-=</code>	<code>a -= 3</code>	<code>a = a - 3</code> 과 동일
<code>*=</code>	<code>a *= 3</code>	<code>a = a * 3</code> 과 동일
<code>/=</code>	<code>a /= 3</code>	<code>a = a / 3</code> 과 동일
<code>//=</code>	<code>a //= 3</code>	<code>a = a // 3</code> 과 동일
<code>%=</code>	<code>a %= 3</code>	<code>a = a % 3</code> 과 동일
<code>**=</code>	<code>a **= 3</code>	<code>a = a ** 3</code> 과 동일

Section 01 파이썬의 기초 문법

■ 관계 연산자(표 2-4)

개념	연산자	의미	설명
$a < b = \begin{cases} \text{참} : \text{True} \\ \text{거짓} : \text{False} \end{cases}$	<code>==</code>	같다.	두 값이 동일하면 참
	<code>!=</code>	같지 않다.	두 값이 다르면 참
	<code>></code>	크다.	왼쪽이 크면 참
	<code><</code>	작다.	왼쪽이 작으면 참
	<code>>=</code>	크거나 같다.	왼쪽이 크거나 같으면 참
	<code><=</code>	작거나 같다.	왼쪽이 작거나 같으면 참

```
a, b = 100, 200  
print(a == b, a != b, a > b, a < b, a >= b, a <= b)
```

실행 결과

False True False True False True

`a==b`는 100이 200과 같다는
의미이므로 결과는 거짓(False)

Section 01 파이썬의 기초 문법

- 문자열과 숫자의 상호 변환
 - 숫자로 구성된 문자열은 int() 또는 float() 함수를 사용하여 정수나 실수로 변환할 수 있음

[illegible]

실행 결과

101 101.123 10000000000000000000000000000000

```
a = 100; b = 100.123
str(a) + '1'; str(b) + '1'
```

실행 결과

```
'1001'
```

```
'100.1231'
```

■ 조건문

■ 기본 if 문

[조건문과 반복문 동영상 강좌](#)

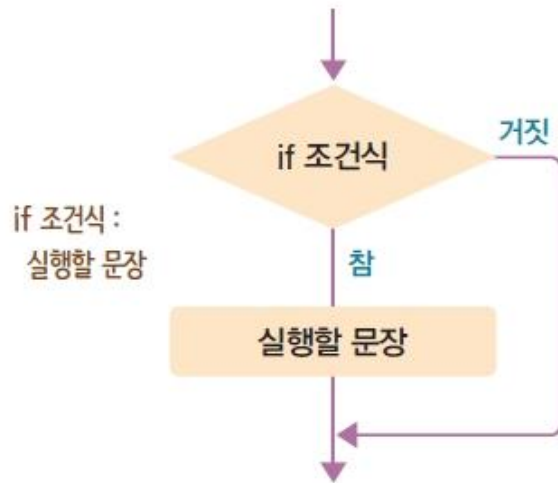


그림 2-1 if 문의 형식과 순서도

```
a = 99
if a < 100 :
    print("100보다 작군요.")
```

실행 결과

100보다 작군요.

Section 01 파이썬의 기초 문법

▪ if~else 문

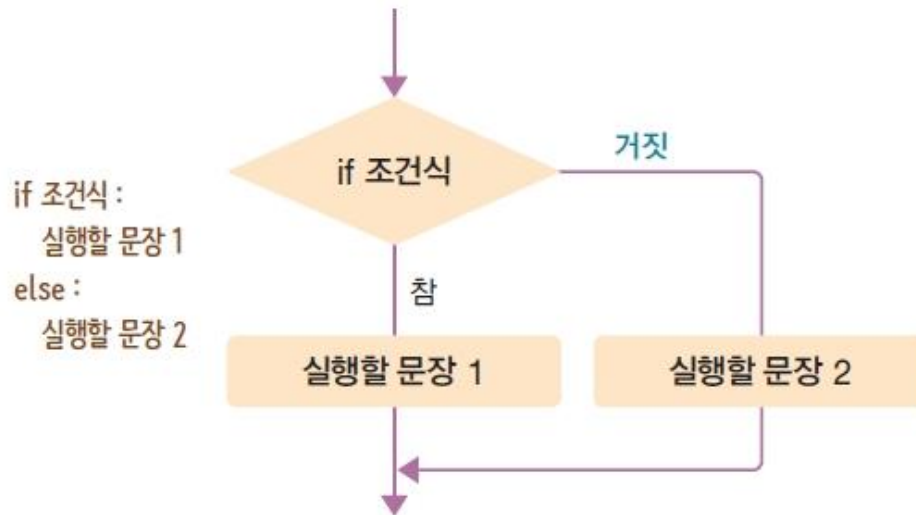


그림 2-2 if~else 문의 형식과 순서도

```
a = 200
```

```
if a < 100 :
```

```
    ❶ print("100보다 작군요.")
```

```
else :
```

```
    ❷ print("100보다 크군요.")
```

a에 200이 들어 있으므로 a<100의 조건식은 거짓이 되어 else 아래에 있는 ❷를 실행

실행 결과

100보다 크군요.

Section 01 파이썬의 기초 문법



여기서 잠깐

C, 자바 등은 중괄호({ })를 사용하여 프로그램의 블록(block)을 지정하지만, 파이썬은 블록으로 묶을 수 있는 기호가 별도로 없어 콜론(:)과 들여쓰기로 지정한다. 콜론 이후에 나란히 들여 쓴 행까지가 하나의 블록이 된다. 블록이 필요한 구문으로는 if, for, while, def 등이 있다.

파이썬의 블록

Section 01 파이썬의 기초 문법

■ 반복문

■ for 문 형식과 사용 예

```
for 변수 in range(시작값, 끝값+1, 증가값) :  
    이 부분을 반복
```

```
for i in range(0, 3, 1) :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

↓ range(0, 3, 1)은 [0, 1, 2]와 같으므로 내부적으로 다음과 같이 변경

```
for i in [ 0, 1, 2 ] :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```


Section 01 파이썬의 기초 문법

▪ while 문

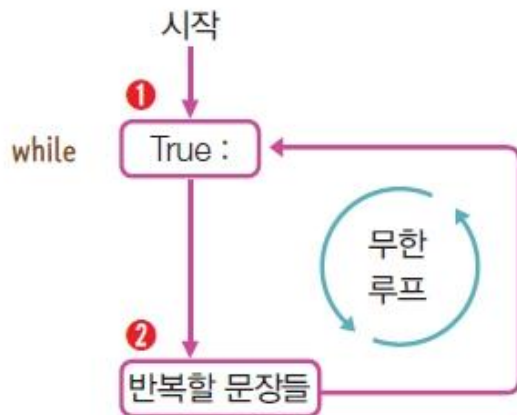


그림 2-3 while 문의 형식과 무한 루프

```
while True :  
    print("=", end = " ")
```

실행 결과

= = = = = ~~~~~ 무한 반복

무한정 글자의 출력을 중지하려면 [ctrl]+[c]를 누름

Section 01 파이썬의 기초 문법

▪ break 문

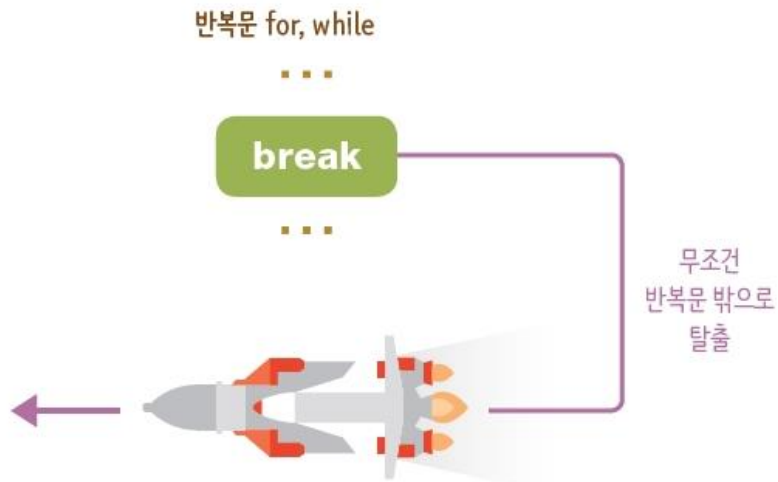


그림 2-4 break 문의 작동

```
for i in range(1, 100) :  
    print("for 문을 %d번 실행했습니다." % i)  
    break
```

for 문이 99번 실행되도록 계획했지만
break 문을 만나 겨우 한 번만 실행하고 종료함

실행 결과

for 문을 1번 실행했습니다.

Section 01 파이썬의 기초 문법

▪ continue 문



그림 2-5 continue 문의 작동

Section 01 파이썬의 기초 문법

■ 함수

■ 함수의 형식과 예

함수이름()

```
print("IT@COOKBOOK for Beginner")
```

괄호 안에 들어 있는 내용을 화면에 출력하는 함수



함수는 매개변수를 입력받은 후
그 매개변수를 가공 및 처리해서 변환
값을 돌려줌

그림 2-6 함수의 기본 형식

Section 01 파이썬의 기초 문법

Code02-01.py

```
1  ## 함수 선언 부분 ##
2  def plus(v1, v2) :
3      result = 0
4      result = v1 + v2
5      return result
6
7  ## 전역 변수 선언 부분 ##
8  hap = 0
9
10 ## 메인 코드 부분 ##
11 hap = plus(100, 200)
12 print("100과 200의 plus() 함수 결과는 %d" % hap)
```

실행 결과

100과 200의 plus() 함수 결과는 300

Section 01 파이썬의 기초 문법

여기서 잠깐



코드의
가독성을
높이는 구성

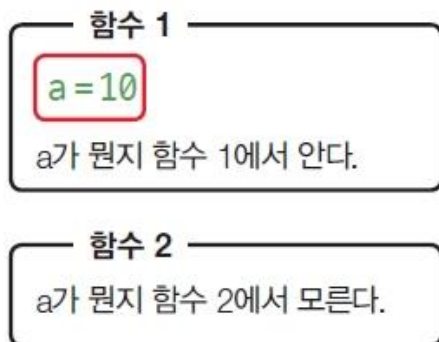
파이썬은 별도의 변수 선언이나 메인 함수가 없어 코드가 길어지면 가독성이 떨어진다. 그래서 필자는 함수 등으로 코드가 길어지면 함수 선언 부분, 전역 변수 부분, 메인 코드 부분으로 분할해서 코드를 작성했다. 필수 사항은 아니지만, 다른 프로그래밍 언어와 형태가 비슷하며 코드를 읽기가 좀 더 수월하다.

Section 01 파이썬의 기초 문법

■ 지역 변수와 전역 변수

[함수와 지역변수 동영상 강좌](#)

❶ 지역 변수의 생존 범위



❷ 전역 변수의 생존 범위

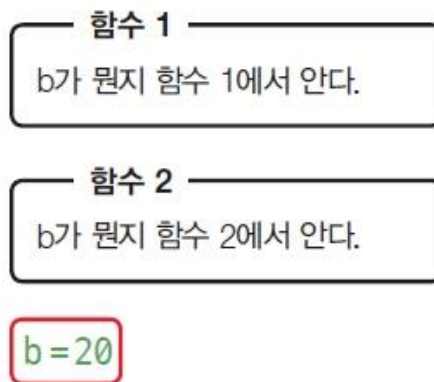


그림 2-7 지역 변수와 전역 변수의 생존 범위

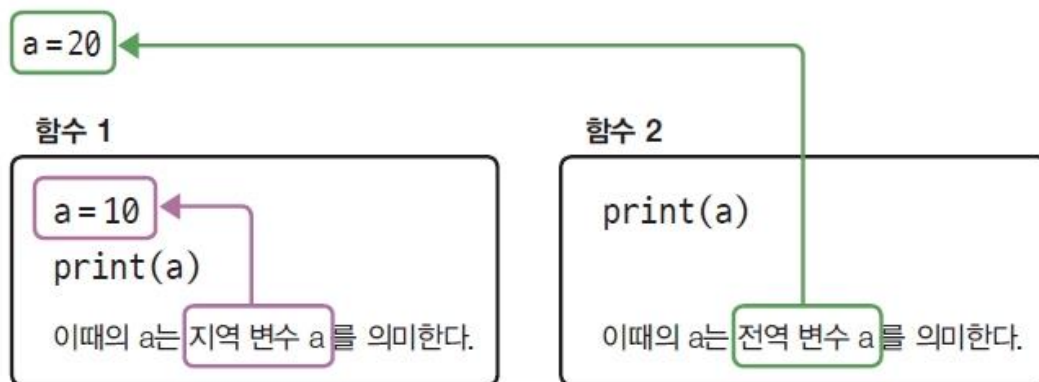


그림 2-8 지역 변수와 전역 변수의 공존

Section 01 파이썬의 기초 문법



여기서 잠깐

C, 자바, C# 등은 main() 함수가 필수로 있어 프로그램 시작 위치가 명확하지만, 파이썬은 필수가 아니다. 하지만 필요하다면 비슷한 효과를 다음과 같이 표현할 수 있다. Code02-01.py의 11~12행을 다음과 같이 사용해도 된다. 3장부터는 프로그램 시작 위치를 명확하게 하고자 코드가 좀 길어질 때는 다음 형식으로 프로그램을 작성하겠다.

파이썬에서
main() 함수

```
if __name__ == "__main__":  
    hap = plus(100, 200)  
    print("100과 200의 plus() 함수 결과는 %d" % hap)
```


Section 01 파이썬의 기초 문법

Code02-02.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      a = 10    # 지역 변수
4      print("func1()에서 a 값 %d" % a)
5
6  def func2() :
7      print("func2()에서 a 값 %d" % a)
8
9  ## 전역 변수 선언 부분 ##
10 a = 20        # 전역 변수
11
12 ## 메인 코드 부분 ##
13 func1()
14 func2()
```

실행 결과

func1()에서 a 값 10
func2()에서 a 값 20

Section 01 파이썬의 기초 문법

- global 예약어

Code02-03.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      global a      # 이 함수 안에서 a는 전역 변수
4      a = 10
5      print("func1()에서 a 값 %d" % a)
6
7  def func2() :
8      print("func2()에서 a 값 %d" % a)
9
10 ## 함수 변수 선언 부분 ##
11 a = 20            # 전역 변수
12
13 ## 메인 코드 부분 ##
14 func1()
15 func2()
```

실행 결과

func1()에서 a 값 10
func2()에서 a 값 10

Section 01 파이썬의 기초 문법

- 반환 값이 여러 개인 함수

Code02-04.py

```
1  ## 함수 선언 부분 ##
2  def multi(v1, v2) :
3      retList = []  # 반환할 리스트
4      res1 = v1 + v2
5      res2 = v1 - v2
6      retList.append(res1)
7      retList.append(res2)
8      return retList
9
10 ## 전역 변수 선언 부분 ##
11 myList = []
12 hap, sub = 0, 0
13
14 ## 메인 코드 부분 ##
15 myList = multi(100, 200)
16 hap = myList[0]
17 sub = myList[1]
18 print("multi()에서 반환한 값 ==> %d, %d" % (hap, sub))
```

실행 결과

multi()에서 반환한 값 ==> 300, -100

Section 01 파이썬의 기초 문법

SELF STUDY 2-1

100과 20의 더하기, 빼기, 곱하기, 몫, 나머지, 제곱이 출력되도록 프로그램을 구현하자.

실행 결과

`multi()`에서 반환한 값 ==> [120, 80, 2000, 5, 0, 1000000000000000000000000000000000000]

■ 파이썬 데이터 형식의 분류

[파이썬 데이터 형식 동영상 강좌](#)

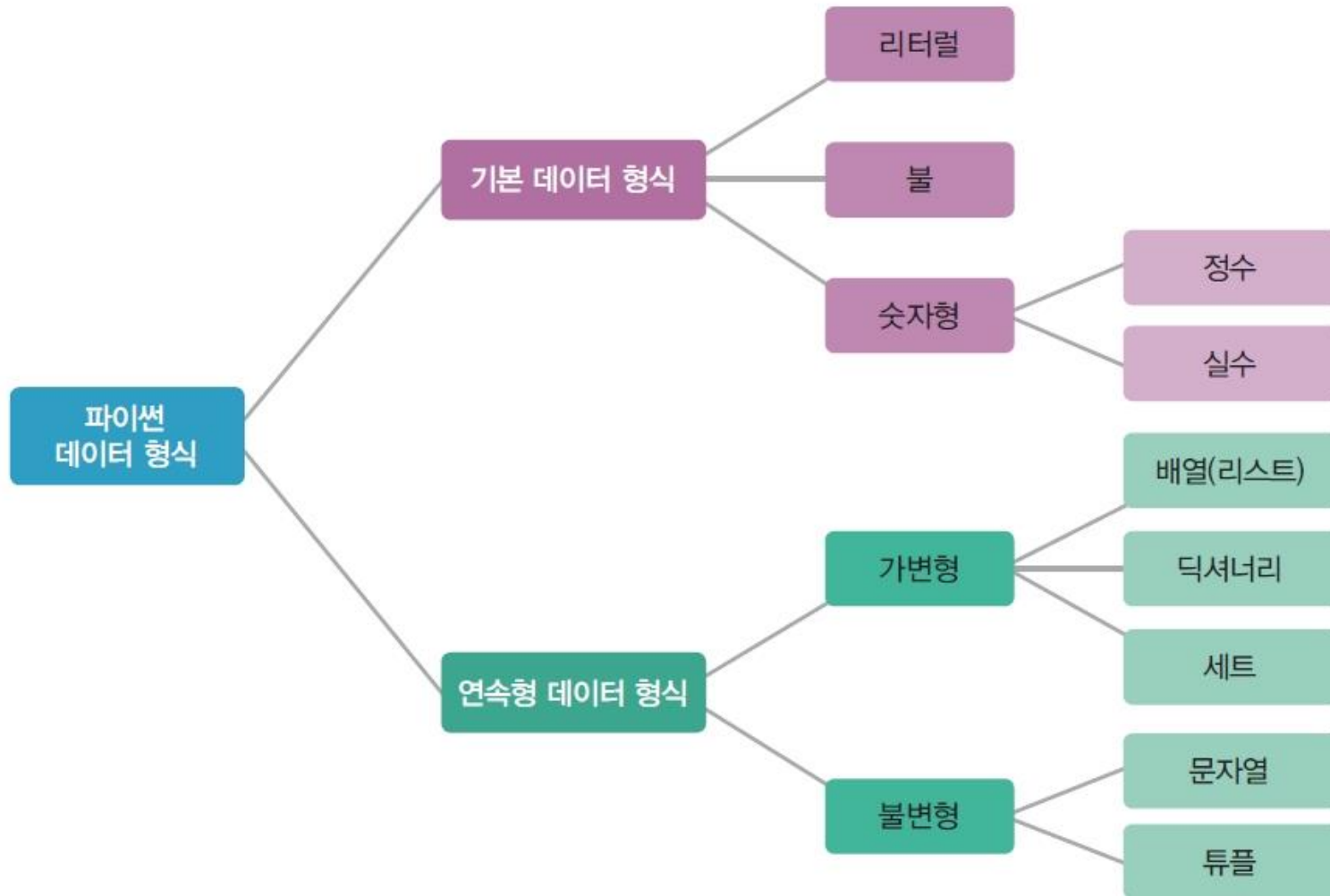


그림 2-9 파이썬 데이터 형식의 분류

Section 02 파이썬의 데이터형

■ 기본 데이터형

- 리터럴(Literal)은 100, 1.23, "Python"처럼 숫자, 문자열 등 상수 값을 직접 표기한 데이터
- 불(bool)은 참이나 거짓을 저장할 수 있고, if 문이나 while 문의 조건식 결과로 사용

```
boolVar = 100 > 200
boolVar
type(boolVar)
```

실행 결과

```
False
<class 'bool'>
```

- 숫자형에는 정수(int)와 실수(float) 두 가지가 있음

```
intVar = 100 ** 100
intVar
type(intVar)
```

실행 결과

```
1000000000000~::~~0000000000  
<class 'int'>
```

Section 02 파이썬의 데이터형

■ 기본 데이터형

- 숫자형에는 정수(int)와 실수(float) 두 가지가 있음

```
intVar = 100 ** 100
intVar
type(intVar)
```

실행 결과

```
1000000000000~::~~000000000000  
<class 'int'>
```

```
floatVar = 0.123456789012345678901234567890
floatVar
type(floatVar)
```

실행 결과

```
0.12345678901234568
<class 'float'>
```

■ 리스트(가변형 데이터 형식)

■ 리스트 개념

- 변수를 한 줄로 붙인 후 전체에 이름(aa) 지정
- 각각은 aa[0], aa[1], aa[2], aa[3]처럼 번호(첨자)를 붙여 사용

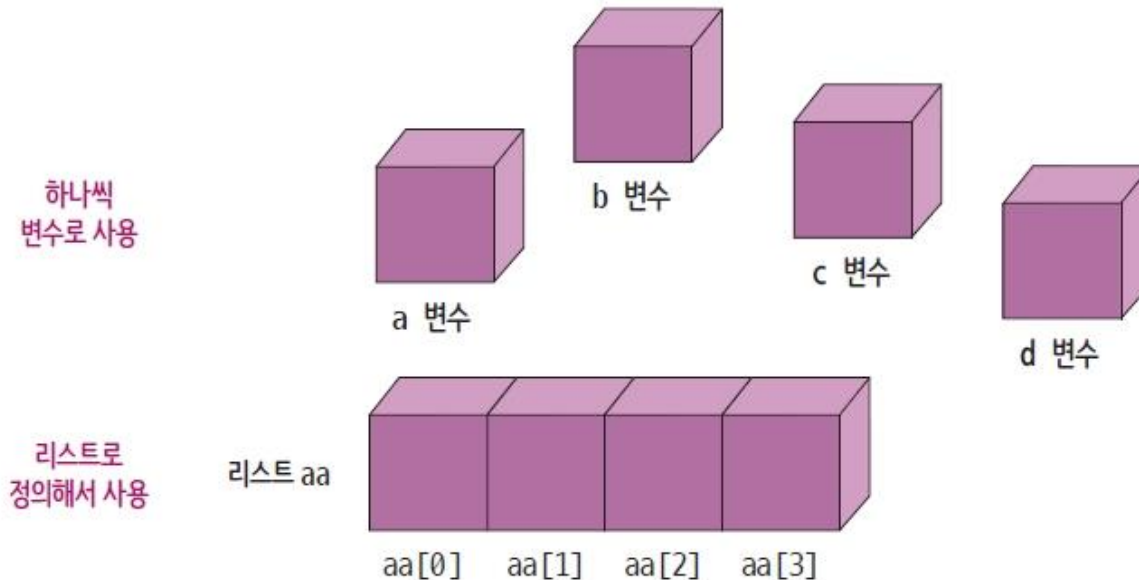


그림 2-10 리스트의 개념

Section 02 파이썬의 데이터형

- 리스트 생성 형식과 예

```
리스트이름 = [값1, 값2, 값3, ...]
```

```
aa = [10, 20, 30, 40]
```

값 4개를 담은 정수형 리스트를 생성함

```
print(aa[0])
```

```
aa[1] = 200
```

리스트는 첨자를 사용하여 **aa[0]** 형식으로 접근

Section 02 파이썬의 데이터형

빈 리스트 생성과 항목 추가

```
aa = []  
aa.append(0)  
aa.append(0)  
aa.append(0)  
aa.append(0)  
print(aa)
```

실행 결과

[0, 0, 0, 0]

```
aa = []  
for i in range(0, 100):  
    aa.append(0)  
len(aa)
```

항목이 100개인 리스트를 만들 경우 `append()` 함수와 함께 `for` 문을 활용

실행 결과

100

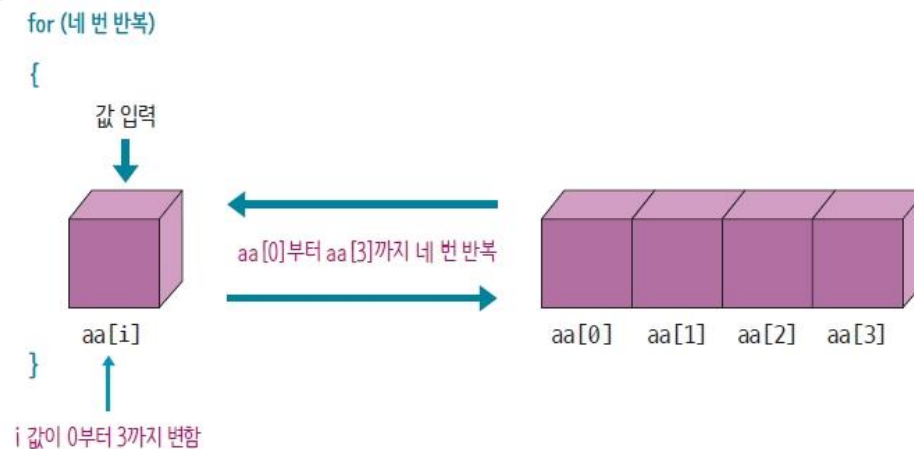


그림 2-11 for 문으로 리스트 값 입력

Section 02 파이썬의 데이터형

Code02-05.py

```
1 aa = []
2 for i in range(0, 4) :
3     aa.append(0)
4 hap = 0
5
6 for i in range(0, 4) :
7     aa[i] = int(input(str(i + 1) + "번째 숫자 : "))
8 hap = 0
9 for i in range(0, 4) :
10     hap = hap + aa[i]
11
12 print("합계 ==> %d" % hap)
```

실행 결과

1번째 숫자 : 10
2번째 숫자 : 20
3번째 숫자 : 30
4번째 숫자 : 40
합계 ==> 100

Section 02 파이썬의 데이터형

- 리스트 값에 접근하는 방법
 - 맨 뒤를 -1 값으로 지정한 첨자로 확인

```
aa = [10, 20, 30, 40]
print("aa[-1]은 %d, aa[-2]는 %d" % (aa[-1], aa[-2]))
```

실행 결과

aa[-1]은 40, aa[-2]는 30

- 콜론(:)을 사용하여 범위 지정. '리스트이름[시작값:끝값+1]'은 리스트 시작 위치부터 끝 위치까지 모든 값을 의미

```
aa = [10, 20, 30, 40]
aa[0:3]
aa[2:4]
```

실행 결과

[10, 20, 30]

[30, 40]

- 콜론의 앞이나 뒤 숫자를 생략할 수 있음

```
aa = [10, 20, 30, 40]
aa[2:]
aa[:2]
```

실행 결과

[30, 40]

[10, 20]

Section 02 파이썬의 데이터형

■ 리스트 조작 함수(표 2-5)

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트이름.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목을 삭제한다).	리스트이름.pop()
sort()	리스트의 항목을 정렬한다.	리스트이름.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트이름.reverse()
index()	지정한 값을 찾아 위치를 반환한다.	리스트이름.index(찾을값)
insert()	지정한 위치에 값을 삽입한다.	리스트이름.insert(위치, 값)
remove()	리스트에서 지정한 값을 제거한다. 단 지정한 값이 여러 개이면 첫 번째 값만 지운다.	리스트이름.remove(지울값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다.	리스트이름.extend(추가할리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트이름.count(찾을값)
clear()	리스트 내용을 모두 지운다.	리스트이름.clear()
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트이름[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트이름)
copy()	리스트 내용을 새로운 리스트에 복사한다.	새리스트 = 리스트이름.copy()
sorted()	리스트의 항목을 정렬해서 새로운 리스트에 대입한다.	새리스트 = sorted(리스트)

Section 02 파이썬의 데이터형

Code02-06.py

```
1  myList = [30, 10, 20]
2  print("현재 리스트 : %s" % myList)
3
4  myList.append(40)
5  print("append(40) 후의 리스트 : %s" % myList)
6
7  print("pop()으로 추출한 값 : %s" % myList.pop())
8  print("pop() 후의 리스트 : %s" % myList)
9
10 myList.sort()
11 print("sort() 후의 리스트 : %s" % myList)
12
13 myList.reverse()
14 print("reverse() 후의 리스트 : %s" % myList)
15
16 print("20 값의 위치 : %d" % myList.index(20))
17
18 myList.insert(2, 222)
19 print("insert(2, 222) 후의 리스트 : %s" % myList)
20 myList.remove(222)
21 print("remove(222) 후의 리스트 : %s" % myList)
22
23 myList.extend([77, 88, 77])
24 print("extend([77, 88, 77]) 후의 리스트 : %s" % myList)
25
26 print("77 값의 개수 : %d" % myList.count(77))
```

실행 결과

현재 리스트 : [30, 10, 20]
append(40) 후의 리스트 : [30, 10, 20, 40]
pop()으로 추출한 값 : 40
pop() 후의 리스트 : [30, 10, 20]
sort() 후의 리스트 : [10, 20, 30]
reverse() 후의 리스트 : [30, 20, 10]
20 값의 위치 : 1
insert(2, 222) 후의 리스트 : [30, 20, 222, 10]
remove(222) 후의 리스트 : [30, 20, 10]
extend([77, 88, 77]) 후의 리스트 : [30, 20, 10, 77, 88, 77]
77 값의 개수 : 2

Section 02 파이썬의 데이터형

- 2차원 리스트 : 1차원 리스트를 여러 개 연결한 것으로 첨자를 2개 사용

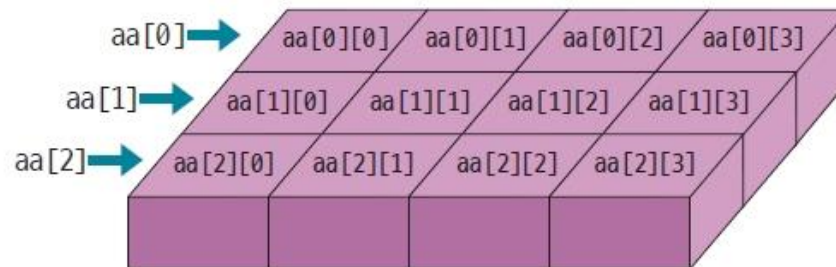
[2차원 리스트와 딕셔너리 동영상 강좌](#)

```
aa = [10, 20, 30]
```



그림 2-12 1차원 리스트의 개념

```
aa = [[1, 2, 3, 4],  
      [5, 6, 7, 8],  
      [9, 10, 11, 12]]
```



전체 리스트 이름 : aa

그림 2-13 2차원 리스트의 개념

Section 02 파이썬의 데이터형

Code02-07.py

```
1 list1 = []
2 list2 = []
3 value = 1
4 for i in range(0, 3):
5     for k in range(0, 4):
6         list1.append(value)
7         value += 1
8     list2.append(list1)
9     list1 = []
10
11 for i in range(0, 3):
12     for k in range(0, 4):
13         print("%3d" % list2[i][k], end = " ")
14     print(" ")
```

실행 결과

```
1  2  3  4
5  6  7  8
9 10 11 12
```


Section 02 파이썬의 데이터형

SELF STUDY 2-2

4행 3열짜리 리스트를 생성한 후 12~1을 입력하고 출력해 보자. 추가로 배열의 모든 숫자 합계도 계산해 보자.

실행 결과

```
12 11 10
```

```
9  8  7
```

```
6  5  4
```

```
3  2  1
```

```
배열의 합계 ==> 78
```

Section 02 파이썬의 데이터형

- 컴프리헨션(함축) 개념
 - 값이 순차적인 리스트를 한 줄로 만드는 간단한 방법

```
numList = []  
for num in range(1, 6) :  
    numList.append(num)  
numList
```

실행 결과

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

↓ 컴프리헨션으로 한 행으로 작성

```
numList = [num for num in range(1, 6)]  
numList
```

실행 결과

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

Section 02 파이썬의 데이터형

▪ 컴프리헨션(함축) 형식

```
리스트 = [수식 for 항목 in range() if 조건식]
```

```
numList = [num * num for num in range(1, 6)]  
numList
```

1~5의 제곱으로 구성된 리스트

실행 결과

```
[1, 4, 9, 16, 25]
```

```
numList = [num for num in range(1, 21) if num % 3 == 0]  
numList
```

it 문 추가하여 1~20 숫자 중
3의 배수로만 리스트 구성

실행 결과

```
[3, 6, 9, 12, 15, 18]
```

```
리스트 = [[수식 for 항목 in range()] for 항목 in range()]
```

 2차원 리스트의 컴프리헨션 구성 형식

```
list2 = [[0 for _ in range(4)] for _ in range(3)]  
list2
```

3x4 크기의 0이 채워진 2차원 리스트 구성 방법

실행 결과

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Section 02 파이썬의 데이터형

■ 딕셔너리(가변형 데이터 형식)

■ 딕셔너리의 개념

- 2개가 하나로 묶인 자료구조로 중괄호 {}로 묶어 구성하며 키와 값으로 구성됨

```
딕셔너리변수 = {키1:값1, 키2:값2, 키3:값3, ...}
```

■ 딕셔너리 생성 예

```
dic1 = {1 : 'a', 2 : 'b', 3 : 'c'}  
dic1
```

키를 1, 2, 3으로 하고 값을 'a', 'b', 'c'로 만듦

실행 결과

```
{1 : 'a', 2 : 'b', 3 : 'c'}
```

```
dic2 = {'a': 1, 'b': 2, 'c': 3}  
dic2
```

키와 값을 반대로 생성함

실행 결과

```
{'a': 1, 'b': 2, 'c': 3}
```

※ 딕셔너리에는 순서가 없어 생성한 순서대로 딕셔너리가 구성되어 있다는 보장을 할 수 없음

Section 02 파이썬의 데이터형

- 딕셔너리는 여러 정보를 변수 하나로 표현할 때 유용

표 2-6 홍길동 학생의 정보

키	값
학번	1000
이름	홍길동
학과	컴퓨터학과



```
student1 = {'학번' : 1000, '이름' : '홍길동', '학과' : '컴퓨터학과'}  
student1
```

실행 결과

```
{'학번' : 1000, '이름' : '홍길동', '학과' : '컴퓨터학과'}
```

```
student1['연락처'] = '010-1111-2222'  
student1
```

student1 연락처를 추가

실행 결과

```
{'학번' : 1000, '이름' : '홍길동', '학과' : '컴퓨터학과', '연락처' : '010-1111-2222'}
```

```
student1['학과'] = '파이썬학과'  
student1
```

이미 존재하는 키를 사용하면 기존 값이 변경됨

실행 결과

```
{'학번' : 1000, '이름' : '홍길동', '학과' : '파이썬학과', '연락처' : '010-1111-2222'}
```

Section 02 파이썬의 데이터형

```
del(student1['학과'])
```

student1의 학과 삭제

```
student1
```

실행 결과

```
{'학번' : 1000, '이름' : '홍길동', '연락처' : '010-1111-2222'}
```

```
student1 = {'학번' : 1000, '이름' : '홍길동', '학과' : '파이썬학과', '학번' : 2000}
```

```
student1
```

동일한 키를 갖는 딕셔너리를 생성하는 것이 아니라 마지막에 있는 키가 적용됨

실행 결과

```
{'학번' : 2000, '이름' : '홍길동', '학과' : '파이썬학과'}
```

※ 딕셔너리 특성상 키는 유일해야 함

Section 02 파이썬의 데이터형

- 딕셔너리 값 접근
 - 키를 사용하여 값을 구함

```
student1['학번']  
student1['이름']  
student1['학과']
```

실행 결과

```
2000  
'홍길동'  
'파이썬학과'
```

딕셔너리이름[키]와 딕셔너리이름.get(키)는 결과가 같음

```
student1.get('이름')
```

실행 결과

```
'홍길동'
```

```
student1['주소']
```

딕셔너리이름[키]는 없는 키를 호출하면 오류 발생

실행 결과

```
오류가 발생
```

딕셔너리이름.get(키)는 없는 키를 호출하면 아무것도 반환하지 않음

Section 02 파이썬의 데이터형

```
student1.keys()
```

딕셔너리이름.**keys()**는 모든 키 반환

실행 결과

```
dict_keys(['학번', '이름', '학과'])
```

```
list(student1.keys())
```

실행 결과의 **dict_keys**를 없애려면 **list(딕셔너리이름.keys())** 함수 사용

실행 결과

```
['학번', '이름', '학과', '연락처']
```

```
student1.values()
```

딕셔너리이름.**values()** 함수는 딕셔너리의 모든 값을 리스트로 만들어 반환

실행 결과

```
dict_values([2000, '홍길동', '파이썬학과'])
```

```
student1.items()
```

딕셔너리이름.**items()** 함수를 사용하면 튜플 형태도 구할 수 있음

실행 결과

```
dict_items([('학번', 2000), ('이름', '홍길동'), ('학과', '파이썬학과')])
```


Section 02 파이썬의 데이터형

```
'이름' in student1  
'주소' in student1
```

딕셔너리에 키가 있다면 **True**를, 없다면 **False**를 반환

실행 결과

True

False

Code02-08.py

```
1 singer = {}  
2  
3 singer['이름'] = '트와이스'  
4 singer['구성원 수'] = 9  
5 singer['데뷔'] = '서바이벌 식스틴'  
6 singer['대표곡'] = 'SIGNAL'  
7  
8 for k in singer.keys():  
9     print('%s --> %s' % (k, singer[k]))
```

for 문을 활용하여 딕셔너리의 모든 값 출력

실행 결과

이름 --> 트와이스

구성원 수 --> 9

데뷔 --> 서바이벌 식스틴

대표곡 --> SIGNAL

Section 02 파이썬의 데이터형

■ 세트(가변형 데이터 형식)

[세트와 문자열 동영상 강좌](#)

- 세트는 키만 모아 놓은 딕셔너리의 특수한 형태
- 딕셔너리의 키는 중복되면 안 되므로 세트에 들어 있는 값은 항상 유일
- 세트를 생성하려면 중괄호 { }를 사용하지만 : 없이 값 입력

```
mySet1 = {1, 2, 3, 3, 3, 4}
mySet1
```

실행 결과

```
{1, 2, 3, 4}
```

중복된 키는 자동으로 하나만 남음

↓ 종류만 파악하고 싶을 때 리스트를 세트로 변경

```
salesList = ['삼각김밥', '바나나', '도시락', '삼각김밥', '삼각김밥', '도시락', '삼각김밥']
set(salesList)
```

실행 결과

```
{'도시락', '바나나', '삼각김밥'}
```

Section 02 파이썬의 데이터형

- 두 세트 사이의 교집합, 합집합, 차집합, 대칭 차집합을 구할 수 있음

```
mySet1 = {1, 2, 3, 4, 5}
```

```
mySet2 = {4, 5, 6, 7}
```

```
mySet1 & mySet2    # 교집합
```

```
mySet1 | mySet2    # 합집합
```

```
mySet1 - mySet2    # 차집합
```

```
mySet1 ^ mySet2    # 대칭 차집합
```

각 연산자 대신 함수 사용 가능

```
mySet1.intersection(mySet2)    # 교집합
```

```
mySet1.union(mySet2)           # 합집합
```

```
mySet1.difference(mySet2)      # 차집합
```

```
mySet1.symmetric_difference(mySet2)    # 대칭 차집합
```

실행 결과

```
{4, 5}
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
{1, 2, 3}
```

```
{1, 2, 3, 6, 7}
```

Section 02 파이썬의 데이터형

■ 문자열(불변형 데이터 형식)

- 문자열은 문자를 연속해서 저장해 놓은 데이터 형식으로 큰따옴표(" ")나 작은따옴표(' ')로 묶어서 표현
- 문자열은 한번 데이터를 저장해 놓으면 변경할 수 없는 불변형 데이터 형식

■ 문자열 기본

```
ss = "자료구조&알고리즘"  
ss[0]  
ss[1:4]  
ss[4:]
```

실행 결과

```
'자'  
'료구조'  
'&알고리즘'
```

Section 02 파이썬의 데이터형

```
ss = '파이썬' + '최고'
```

```
ss
```

```
ss = '파이썬' * 3
```

```
ss
```

문자열도 리스트와 마찬가지로 더하기(+) 기호를 사용하여 연결
곱하기(*) 기호를 사용하여 문자열을 반복할 수 있음

실행 결과

```
'파이썬최고'
```

```
'파이썬파이썬파이썬'
```

```
ss = '파이썬abcd'
```

```
len(ss)
```

문자열 길이를 파악할 때도 리스트처럼 len() 함수 사용

실행 결과

```
7
```

Section 02 파이썬의 데이터형

▪ 문자열 검색

- count('찾을문자열') 함수는 '찾을문자열'이 몇 개 들어 있는지 개수를 셈
- find('찾을문자열') 함수는 '찾을문자열'이 왼쪽 끝(0번 위치)부터 시작해서 몇 번째에 위치하는지 찾음
- find() 함수는 찾을 문자열이 없으면 -1을 반환함
- rfind('찾을문자열') 함수는 find()와 반대로 오른쪽부터 찾음
- index() 함수도 find() 함수와 동일하지만 index('없다')처럼 찾을 문자열이 없으면 오류 발생

```
ss = '파이썬 공부는 즐겁습니다. 물론 모든 공부가 다 재미있지는 않죠. ^^'
ss.count('공부')
print(ss.find('공부'), ss.rfind('공부'), ss.find('공부', 5), ss.find('없다'))
print(ss.index('공부'), ss.rindex('공부'), ss.index('공부', 5))
print(ss.startswith('파이썬'), ss.startswith('파이썬', 10), ss.endswith('^^'))
```

실행 결과

```
2
4 21 21 -1
4 21 21
True False True
```

Section 02 파이썬의 데이터형

▪ 문자열 분리와 결합

- `split()` 함수는 문자열을 공백이나 다른 문자로 분리하여 리스트를 반환함
- `join()` 함수는 문자열을 서로 합침

```
ss = 'Python을 열심히 공부 중'
ss.split()
ss = '하나:둘:셋'
ss.split(':')
ss = '하나\n둘\n셋'
ss.splitlines()
ss = '%'
ss.join('파이썬')
```

실행 결과

```
['Python을', '열심히', '공부', '중']
['하나', '둘', '셋']
['하나', '둘', '셋']
'파%이%썬'
```

Section 02 파이썬의 데이터형

- 함수 이름 대입

- map(함수이름, 리스트이름) 함수는 리스트의 문자열 하나하나를 함수 이름에 대입
- 다음 예와 같은 방법으로 문자열로 구성된 리스트를 숫자로 변환할 수 있음

```
before = ['2022', '12', '31']  
after = list(map(int , before))  
after
```

실행 결과

```
[2022, 12, 31]
```


Section 02 파이썬의 데이터형

■ 튜플(불변형 데이터 형식)

- 리스트와 사용법이 비슷하면서 약간 다름
 - 리스트는 대괄호 []로 생성하지만 튜플은 소괄호 ()로 생성
 - 튜플은 값을 수정할 수 없으며, 읽기만 가능하기에 읽기 전용 자료를 저장할 때 사용
-
- 튜플 생성과 삭제

```
tt1 = (10, 20, 30); tt1
```

```
tt2 = 10, 20, 30; tt2
```

튜플은 tt2처럼 소괄호 ()를 생략해도 됨

실행 결과

```
(10, 20, 30)
```

```
(10, 20, 30)
```

Section 02 파이썬의 데이터형

```
tt3 = (10); tt3
tt4 = 10; tt4
tt5 = (10,); tt5
tt6 = 10,; tt6
```

tt3과 tt4는 튜플이 아닌 일반 값이 됨

※ 항목이 하나인 튜플은 tt5와 tt6처럼 뒤에 쉼표(,)를 붙여야 함

실행 결과

```
10
10
(10,)
(10,)
```

↓ 튜플은 읽기 전용이므로 다음 코드는 오류 발생

```
tt1.append(40)
tt1[0] = 40
del(tt1[0])
```

```
del(tt1)
del(tt2)
```

튜플 자체는 del() 함수로 삭제할 수 있음

Section 02 파이썬의 데이터형

▪ 튜플 사용

```
tt1 = (10, 20, 30, 40)
```

튜플 항목에 접근할 때는 리스트처럼 '튜플이름[위치]'를 사용

```
tt1[0]
```

```
tt1[0] + tt1[1] + tt1[2]
```

실행 결과

```
10
```

```
60
```

```
tt1[1:3]
```

튜플 범위에 접근하려면 리스트와 마찬가지로 '(시작값:끝값+1)'을 사용

```
tt1[1:]
```

```
tt1[:3]
```

실행 결과

```
(20, 30)
```

```
(20, 30, 40)
```

```
(10, 20, 30)
```

Section 02 파이썬의 데이터형

```
tt2 = ('A', 'B')
```

```
tt1 + tt2
```

튜플의 덧셈 및 곱셈 연산도 가능

```
tt2 * 3
```

실행 결과

```
(10, 20, 30, 40, 'A', 'B')
```

```
('A', 'B', 'A', 'B', 'A', 'B')
```

```
myTuple = (10, 20, 30)
```

```
myList = list(myTuple)
```

```
myList.append(40)
```

튜플 → 리스트 → 튜플로 변환한 예

```
myTuple = tuple(myList)
```

```
myTuple
```

실행 결과

```
(10, 20, 30, 40)
```

응용예제 01 자동 로또 번호 생성기

난이도 ★☆☆☆☆

예제 설명

1~45 숫자 중에서 6개를 고르는 로또 번호를 자동으로 만들어 주는 프로그램을 작성하자. 사용자가 입력한 개수만큼 번호 쌍을 생성해 주는 기능을 한다.



실행 결과

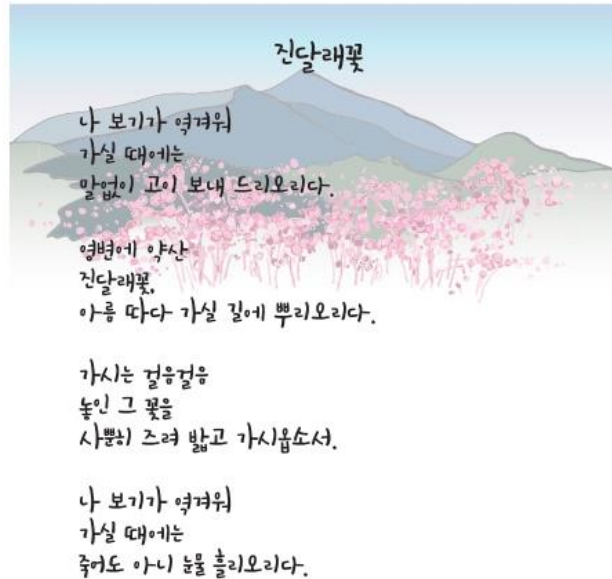
```
Python
File Edit Shell Debug Options Window Help
===== RESTART: C:\WookData\WEx02-01.py =====
** 로또 번호 생성을 시작합니다. **
몇 번을 뽑을까요?5
자동번호--> 14 23 25 27 33 37
자동번호--> 3 13 27 32 33 37
자동번호--> 5 13 14 27 32 37
자동번호--> 4 8 9 13 40 44
자동번호--> 3 9 13 18 27 44
>>>
```

응용예제 02 가장 많이 나온 글자 세기

난이도★★★★☆

예제 설명

김소월의 진달래꽃 시에서 4회 이상 나온 글자를 센다. 단 영문/한글만 해당하고, 그 외는 인정하지 않는다.



실행 결과

```
Python
File Edit Shell Debug Options Window Help
===== RESTART: C:\CookData\Ex02-02.py =====
원문
나 보기가 역겨워 가실 때에는 말없이 고이 보내 드리오리다
영변에 약산 진달래꽃 아름답다 가실 길에 뿌리오리다
가시는 걸음걸음 놓인 그 꽃을 사뿐히 즈려 밟고 가시옵소서
나 보기가 역겨워 가실 때에는 죽어도 아니 눈물 흘리오리다

문자 빈도수(4회 이상)
가 -> 7
에 -> 4
리 -> 6
다 -> 4
>>>|
```



Thank You
