02 자료형

목차

변수 선언하기

◆ 변수란? 어떤 값을 저장할 수 있는 컴퓨터의 물리적 저장 공간인 메모리에 붙이는 이름

- 변수를 이용해 컴퓨터에 데이터를 저장하거나 읽기 가능
- 변수를 만든다 → 숫자나 문자열과 같은 데이터를 저장할 수 있는 공간을 마련하는 것

```
a = 1024      # 변수 a에 1024를 저장
print(a)      # a를 화면에 출력
```

출력결과

```
1024
```

- 변수에 숫자를 할당하면 숫자형, 문자를 할당하면 문자형 변수가 됨

```
a1 = 2048                  # 변수 a1은 숫자를 저장
a2 = "안녕하세요."          # 변수 a2는 문자열을 저장
#a2 = '안녕하세요.'
```

변수명 규칙

- 변수명은 영문자, 숫자, 언더바(_)로 구성
  - 특수문자(&, *, ,(쉼표), %, $, #, @, !)나 공백 등은 사용불가
  - 문자 또는 언더바로 시작해야 하며 숫자로 시작하면 안 됨
  - 파이썬의 예약어(Reserved Words)를 변수로 사용하면 에러 발생

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

  - 변수명 규칙을 어기면 SyntaxError: invalid syntax를 출력

```
100원 = "백원"                    # 변수명을 숫자 100으로 시작하여 에러 발생
email@ = "test@naver.com"         # 변수명에 특수문자 @를 사용하여 에러 발생
----------------------------------------------------출력결과----------------------------------------------------
SyntaxError: invalid syntax
```

상수 사용하기

- 상수(constant)란? 변하지 않는 값 혹은 데이터
  - 숫자형 상수는 있는 그대로 표시
  - 문자나 문자열 상수는 작은따옴표(') 혹은 큰따옴표(")로 표현

```
print(20)                    # 상수 20을 화면에 출력
print('A')                   # 문자 상수 'A'를 화면에 출력
print("선생님")              # 문자열 상수 "선생님"을 화면에 출력
----------------------------------------------------출력결과----------------------------------------------------
20
A
선생님
```

  - 변수와 상수는 **같은 유형끼리** 연산 가능

```
s = "선생님"                                  # 변수 s에 "선생님" 저장
s = s + ", " + "안녕하세요!"                  # 변수 s에 ", "와 "안녕하세요!"를 더하여 s에 다시 저장
print(s)                                      # 변수 s를 화면에 출력
----------------------------------------------------출력결과----------------------------------------------------
선생님, 안녕하세요!
```

숫자

● 숫자(Number) 자료형 : 정수, 실수로 구분

➢ 정수형(Integer) : 소수점이 없는 숫자(Digit), 음의 정수도 포함
➢ type() : 자료형을 확인하는 함수

```
a = 359                    # 변수 a에 359를 저장
print(type(a))             # a의 자료형을 출력
----------------------------------------------------출력결과----------------------------------------------------
<class 'int'>              # 변수 a에 저장된 자료가 정수형이라는 것을 알 수 있음

# print() 함수를 사용하지 않고 변수명만 입력해도 변수에 저장된 값을 확인할 수 있다
a
----------------------------------------------------출력결과----------------------------------------------------
359
```

➢ 실수형(Floating-point) : 0.29, -3.704와 같이 소수점이 있는 숫자

```
a = -3.619                 # 변수 a에 -3.619를 저장
print(type(a))             # a의 자료형을 출력
----------------------------------------------------출력결과----------------------------------------------------
<class 'float'>            # 변수 a에 저장된 자료가 정수형이라는 것을 알 수 있음
```

숫자

● 연산자 종류 및 연산에 사용하는 함수

➢ 연산자 종류

연산자	설명	문법	설명
+	더하기(addition)	x + y	x와 y를 더한다.
-	빼기(subtraction)	x - y	x에서 y를 뺀다.
*	곱하기(multiplication)	x * y	x와 y를 곱한다.
/	나누기(division)	x / y	x를 y로 나눈다.
**	**거듭제곱(power)**	x ** y	x를 y번 곱한다(x를 y제곱한다).
//	**몫(quotient)**	x // y	x를 y로 나누었을 때 몫을 구한다.
%	**나머지(remainder)**	x % y	x를 y로 나누었을 때 나머지 값을 구한다.

➢ 연산에 사용하는 함수

문 법	설 명	문 법	설 명
abs(x)	x를 절대값으로 변환	**int(x)**	x를 정수형으로 변환
divmod(x, y)	x를 y로 나눈 몫과 나머지 구하기	**pow(x, y)**	x를 y번 곱하기
float(x)	x를 실수형으로 변환	**round(x, y)**	x를 소수점 y번째 자리까지 반올림

숫자

● 숫자 연산하기

➢ 사칙연산 가능 ※ 사칙연산 : 더하기(+), 빼기(–), 곱하기(*), 나누기

```
a = 27                          # 변수 a에 27을 저장
b = 9                           # 변수 b에 9를 저장
print(a + b, a – b, a * b)      # a와 b 더하기(+), 빼기(–), 곱하기(*) 연산자 사용
```

----------출력결과----------

```
36 18 243
```

➢ 나누기 연산자는 구하는 값에 따라 다른 연산자를 사용 ※ 나누기(/), 몫(//), 나머지(%)

```
print(a/b, a//b, a%b, a**3)     # a를 b로 나누었을 때(실수형), 값(/), 몫(//), 나머지(%), 3제곱 구하기 연산자 사용
```

----------출력결과----------

```
3.0 3 0 19683
```

숫자

● 연산자 종류 및 연산에 사용하는 함수

➢ 함수활용 : 절대값과 반올림 구하기

```
print(abs(-3))                  # -3의 절대값을 구함
print(round(3.14159, 3))        # 실수 3.14159를 소수점 넷째 자리에서 반올림하여 셋째 자리까지 출력
----------------------------------------------------출력결과----------------------------------------------------
3
3.142
```

● 8진수와 16진수

➢ 8진수(Octal) : 0o 또는 0O(숫자 0 뒤에 알파벳 소문자 o 또는 대문자 O) 뒤에 값을 입력해서 표현
➢ 16진수(Hexadecimal) : 0x(숫자 0뒤에 알파벳 소문자 x) 뒤에 값을 입력해서 표현

```
a = 0o467               # 8진수 표현, 10진수로 변환하면 4*8^2 + 6*8^1 + 7*8^0
b = 0x7fc               # 16진수 표현
print(a, b)
----------------------------------------------------출력결과----------------------------------------------------
311 2044
```

주석

➢ **주석(comment)**

- **프로그램에 설명을 달기 위한 용도**
- **한줄 주석 : #**
- **여러줄 주석 : """ """ 변수에 할당하지 않고 처리**
  - ✓ **₩n(줄바꿈이 나타난다)**

주석(comment)

In [3]:
```
# 한줄 주석. 문장의 끝까지 주석 처리됨
```

In [5]:
```
"""
(소스코드를 읽는 사람에게만 의미가 있고 프로그램 실행과는 관련이 없는 문자열)
여러 줄에
걸친
문자열을
나타냅니다
"""
```

Out [5]:
```
'₩n(소스코드를 읽는 사람에게만 의미가 있고 프로그램 실행과는 관련이 없는 문자열)₩n여러 줄에 ₩n걸친 ₩n문자열을₩n나타냅니다₩n'
```

문자열

- **문자열(String) : 문자들의 배열로 이루어진 자료형**
  - ➢ 문자를 입력할 때, 문자열 앞뒤에 작은따옴표(') 또는 큰따옴표(") 사용
  - ➢ 문자열의 타입은 str

```
a = "Hello Python!"          # a = 'Hello Python!'
print(a)
print(type(a))
----------------------------------------------------출력결과----------------------------------------------------
Hello Python!
str
```

문자열

● 문자열 안에 따옴표 포함시키기

➢ 문자열 안에 들어가는 따옴표를 문자열 앞뒤에 붙이는 따옴표와 다른 따옴표로 입력

```
sen = "It's mine."          # 작은따옴표를 문자열 안에 포함할 때
print(sen)
```

----------출력결과----------

```
It's mine.
```

➢ 문자열 앞뒤와 문자열 안에 같은 따옴표를 사용하는 경우, 이스케이프 코드인 ₩(백슬래시) 사용

```
sen2 = "₩"It is impossible. ₩" He says."          # 큰따옴표를 문자열 앞뒤와 안에 사용
print(sen2)
```

----------출력결과----------

```
"It is impossible." He says.
```

문자열

● 문자열을 여러 줄로 나타내기

➢ 작은 따옴표(또는 큰 따옴표)를 3개 연속으로 사용하여 줄바꿈하면 모든 공백과 줄바꿈을 그대로 표현

```
a = """Hello!
Python!"""
print(a)
```

----------출력결과----------

```
Hello!
Python
```

➢ 줄 바꿈 이스케이프 코드(₩n)를 사용하면 같은 문자열 내에서도 줄바꿈 가능

```
a = "Hello!₩nPython!"
print(a)
```

----------출력결과----------

```
Hello!
Python
```

문자열

● 자주 사용하는 escape code

➢ ₩ 뒤에 오는 문자나 숫자에 따라 특정한 기능을 수행하도록 정의해 둔 코드

코드	설명
\\	문자열 안에서 ₩(백 슬래시)를 그대로 표시
\’	문자열 안에서 ‘(작은 따옴표)를 그대로 표시
\”	문자열 안에서 “(큰 따옴표)를 그대로 표시
\b	한 칸 지움. 백 스페이스(backspace)
\f	커서를 다음 줄로 이동. 폼 피드(formfeed)
\n	문자열 안에서 줄 바꿈(linefeed)
\r	줄 바꿈. 캐리지 리턴(carriage return)
\t	문자열 사이의 간격 주기. 탭(tab)

문자열

● 문자열 연산하기

➢ 더하기(+) : 문자열 끼리 **연결**하는 것

➢ 곱하기(*) : 곱한 수만큼 **반복**하는 것

```
a = “Hello! ”
b = “Python!”
print(a+b)                  # 문자열 변수인 a와 b를 연결
a * 3                       # 문자열 변수 a를 3번 반복
----------------------------------------------------출력결과----------------------------------------------------
Hello Python!
Hello! Hello! Hello!
```

➢ **자료형이 다른 유형끼리 연산을 수행하면 TypeError 발생**

문자열

● 문자열 인덱싱

- ➢ 인덱싱(Indexing) : 문자열의 각 문자마다 번호를 매기는 것
- ➢ 표현식 : 문자열[숫자] 또는 변수명[숫자]
- ➢ 첫 문자는 인덱스 [0] 부터 시작
- ➢ 뒤쪽 인덱스 시작 : [-문자열 길이]

```
a = “Hello! Python!”
```

	H	e	l	l	o	!		P	y	t	h	o	n	!
앞쪽 인덱스	0	1	2	3	**4**	5	6	7	8	9	10	11	12	13
뒷쪽 인덱스	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	**-1**

```
print(a[4])          # 앞에서부터 인덱스를 계산, 인덱스 4번 위치는 ‘o’
print(a[-1])         # -로 뒤에서부터 인덱스를 계산, 인덱스 -1번 위치는 !
----------------------------------------------출력결과----------------------------------------------
o
!
```

문자열

● 문자열 슬라이싱

- ➢ 슬라이싱(Slicing) : 인덱싱 번호를 사용해서 원하는 길이만큼 잘라내는 것
- ➢ 표현식 : 문자열[시작번호:끝번호] 또는 변수명[시작번호:끝번호]
- ➢ 끝번호의 **-1번째까지** 잘라냄

```
a = “Hello! Python!”
```

	H	e	l	l	o	!		P	y	t	h	o	n	!
앞쪽 인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13
뒷쪽 인덱스	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
print(a[3 : 9])                    # 인덱스 3번부터 8번까지 추출
print(a[3 : -4])                   # 인덱스 3번부터 -5까지 추출
----------------------------------------------------출력결과----------------------------------------------------
lo! Py
lo! Pyt
```

문자열

● 문자열 슬라이싱

➢ 시작 번호를 비우면 문자열의 **처음부터** 추출, 끝 번호를 비우면 문자열의 **끝까지** 추출

➢ 시작, 끝 번호를 모두 비우면 **문자열 전체**가 추출

```
a = "Hello! Python!"
```

	H	e	l	l	o	!		P	y	t	h	o	n	!
앞쪽 인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13
뒷쪽 인덱스	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
a[:8]                                   # 문자열 처음부터 인덱스 7까지 추출
----------------------------------------출력결과----------------------------------------
'hello! P'
```

```
a[3:]                                   # 인덱스 3부터 문자열 끝까지 추출
----------------------------------------출력결과----------------------------------------
'lo! Python!'
```

```
a[:]                                    # 문자열 전체 추출 → a 와 동일
----------------------------------------출력결과----------------------------------------
'Hello! Python!'
```

문자열

● 슬라이싱을 활용하면 문자열 안의 값을 바꿀 수 있다.

```
a = "Life"                                          # 변수 a에 문자열 자료형 "Life"를 저장
b = a[0:2] + "v" + a[-1]  # 슬라이싱 활용
print(b)
```

----------출력결과----------

Live

문자열

● 문자열 포맷팅

- ➢ 문자열 안에 어떤 값(숫자 또는 문자열 등)을 바꿀 때 사용
- ➢ 변수를 지정해서 변수에 저장된 값으로도 변경 가능
- ➢ 표현식 : "~~ %s ~~" %문자(or 숫자) 또는 "~~ %s ~~" %변수명

```
"I eat %s bananas." %3                  # %s 대신 3 을 대입해서 값을 출력
----------------------------------------출력결과----------------------------------------
'I eat 3 bananas.'

"I eat %s bananas." %"three"            # %s 대신 "three" 를 입력해서 값을 출력
----------------------------------------출력결과----------------------------------------
'I eat three bananas.'

color = "blue"                          # color 변수 생성
" I like %s color." %color              # %s 대신 변수 color 에 저장된 "blue" 값을 출력
----------------------------------------출력결과----------------------------------------
'I like blue color.'
```

문자열

● 문자열 포맷팅

➢ 포맷 코드

코드	설명	입력되는 값
%c	**character**의 첫 글자, **문자 1개**를 의미	문자 1개, 0 또는 양의 정수
%d	**digit**의 첫 글자로 **정수형 숫자**	음의 정수, 0, 양의 정수
%f	**floating point**의 첫 글자로 **실수형 숫자**를 의미	실수(정수 포함)
%s	**string**의 첫 글자, **문자열**을 의미	문자열, 실수, 정수

➢ 문자열의 길이 설정 : 포맷 코드에 입력한 수만큼 길이 지정 및 정렬 가능,
양의 정수는 **오른쪽** 정렬 / **음**의 정수는 **왼쪽** 정렬

```
ar = "%15s" %"hello"                    # 문자열 길이 15, hello 오른쪽 정렬
ar2 = "%-15s" %"hello"                  # 문자열 길이 15, hello 왼쪽 정렬
print(ar)
ar2
```

----------출력결과----------

```
'----------hello'
'hello----------'
```

문자열

● 문자열 함수

➢ format()함수 : “{}”.format(값)

- **% 연산자 대신** 사용.
- 문자열 안에 중괄호({})로 표현하고, 중괄호 안에 format(값) 대입

```
test = “I need {} eggs.”.format(5)      # format 함수에 있는 값 “5”가 { } 위치에 들어감
print(test)
----------------------------------------------------출력결과----------------------------------------------------
I need 5 eggs.
```

- 입력할 값이 여러개인 경우, 값을 변수로 지정한 후 출력. 이 때 변수값은 순서대로 적용됨

```
color = “white”
num = 5
test = “I like {} egg. I need {} eggs.”.format(color, num)          # 여러 개의 값을 변수로 입력
#test = "I like {0} egg. I need {1} eggs.".format(color, num)      # 인덱스 지정. 같은 결과
print(test)
----------------------------------------------------출력결과----------------------------------------------------
I like white egg. I need 5 eggs.
```

문자열

● 문자열 함수

➢ 소수점 자릿수를 지정해 실수 표현하기

: 콜론 앞에는 format 인자의 인덱스 입력, 뒤에는 .숫자f 로 입력

```
a = "{:.2f}".format(5.6182359)          # 소수점 둘째 자리까지 표현(소수점 셋째 자리에서 반올림)
print(a)
```

출력결과

```
5.62
```

```
a = "{0:.2f} / {1:.1f}".format(3.22521321, 10.123456)   # 콜론 앞에는 인덱스 번호, 뒤에는 소수점 자리수 표현
#a = "{:.2f} / {:.1f}".format(3.22521321, 10.123456)    # 인덱스 번호를 생략하면 차례대로 적용
print(a)
```

출력결과

```
3.23 / 10.1
```

```
x = 5.6182359
a = "{:15.2f}".format(x)                # 숫자 길이 15, 오른쪽 정렬, 소수점 둘째 자리까지 표현
print(a)
```

출력결과

```
          5.62
```

문자열

● 문자열 함수

➢ count() 함수

- 문자의 개수

```
a = "I eat banana."
a.count("n")                                    # 문자열에서 “n”의 개수 구하기
----------------------------------------------------출력결과----------------------------------------------------
2
```

➢ join() 함수

- 문자열 안에 값을 삽입

```
b = ",".join("fruit")                           # join() 함수 안에 있는 문자열의 문자 사이에 “,” 삽입하기
print(b)
----------------------------------------------------출력결과----------------------------------------------------
f,r,u,i,t
```

문자열

● 문자열 함수

➢ **find(), index() 함수 : (찾을 문자열, 시작점 [, 종료점])**

- 찾고 싶은 문자의 위치를 찾아 인덱스 값을 반환
- 문자열 안에 찾는 문자가 중복으로 들어 있으면 첫 번째 위치에 있는 문자의 인덱스 값
- **find() : 문자열만, 없으면 -1 반환**
- **index() : 문자열, 리스트, 튜플 가능. 딕셔너리 불가능. 없으면 ValueError**

```
a = "I eat banana."
print(a.find("n"))                  # find() 함수 활용, 문자열에서 “n”이 처음 나오는 인덱스 값 구하기
print(a.find("a", 5))               # 5번째 위치부터 처음 “a”가 위치한 자리
print(a.index("n"))                 # index() 함수 활용, 문자열에서 “n”이 처음 나오는 인덱스 값 구하기
print(a.index("a", 1, 8))           # 1번째~8번째 사이에 처음 “a”가 위치한 자리
----------------------------------------------------출력결과----------------------------------------------------
8
7
8
3
```

문자열

● 문자열 함수

➢ upper(), lower() 함수

- 영문 소문자를 대문자로, 대문자를 소문자로 바꿈

```
a = "Hello"
print(a.upper())                    # 문자를 전부 대문자로 변경
print(a.lower())                    # 문자를 전부 소문자로 변경
----------------------------------------------------출력결과----------------------------------------------------
HELLO
hello
```

➢ strip() 함수

- 문자열 사이의 공백을 지움
- **strip()은 문자열의 양끝(왼쪽, 오른쪽) 공백 제거, lstrip()은 왼쪽 공백, rstrip()은 오른쪽 공백을 제거**

```
a = "  Hello  "
print(a.strip()) ; print(a.lstrip()) ; print(a.rstrip())
----------------------------------------------------출력결과----------------------------------------------------
Hello
Hello--
--Hello
```

문자열

● 문자열 함수

➢ split() 함수

- 문자열을 공백 또는 특정 값으로 분리
- 구분자를 생략하면 공백으로 문자열 분리, 분리된 값은 **list** 자료형으로 저장됨

```
a = “I eat banana.”
a.split()                    # 공백을 기준으로 문자열 나누기
----------------------------------------------------출력결과----------------------------------------------------
['I', 'eat', 'banana.']

a = “a:b:c:d:e”
a.split(“:”)                 # “:”을 기준으로 문자열 나누기
----------------------------------------------------출력결과----------------------------------------------------
['a', 'b', 'c', 'd', 'e']
```

문자열

● 문자열 함수

➢ replace() 함수

- 문자열의 값을 원하는 값으로 변경
- 표현식 : replace(찾을 값, 바꿀 값, 바꿀 횟수) ※ 바꿀 횟수 생략 시 찾은 값 모두 변경

```
a = "I eat banana."
print(a.replace("a", "b", 1))          # 문자열에서 "a"를 "b"로 1회 변경
print(a.replace("a", "b"))             # 문자열에서 "a"를 "b"로 모두 변경
----------------------------------------------------출력결과----------------------------------------------------
I ebt banana.
I ebt bbnbnb.
```

불(Bool)

● 불(Bool) : 참(True) 또는 거짓(False)을 나타내는 자료형

➢ 불린(Boolean)이라고도 부름

➢ 자료형을 판단해서 True 또는 False 중 하나의 값을 나타냄

```
print(1 == 1)          # 1과 1이 같은지 판단, True 출력
print(10 < 8 )         # 10이 8보다 작은지 판단, False 출력
----------------------------------------------------출력결과----------------------------------------------------
True
False
```

값	True / False	설 명	값	True / False	설 명
"python"	True	문자열 자료형	()	False	비어 있는 튜플 자료형
""	False	입력값이 없는 자료	{}	False	비어 있는 딕셔너리 자료형
" "	True	공백(띄어쓰기)	1	True	숫자 자료형
[1, 2, 3]	True	입력값이 있는 리스트 자료형	0	False	숫자 자료형에서 0
[]	False	비어 있는 리스트 자료형	None	False	None은 아무것도 없다는 것을 의미

리스트(list)

● 리스트(list) : 여러 개의 데이터를 [ ]를 이용, 하나의 변수에 담을 수 있는 데이터 구조

➢ 표현식 : 변수명 = [데이터1, 데이터2, 데이터3, ...]

➢ 데이터에는 숫자형, 문자열, 리스트 등 다양한 자료형이 들어갈 수 있음

```
a = [ ]                                  # 공백(띄어쓰기)을 데이터로 갖는 리스트
b = [1, 2, 3, 4, 5]                      # 숫자형을 데이터로 갖는 리스트
c = [“I”, “like”, “python”]              # 문자열을 데이터로 갖는 리스트
d = [1, 2, “Python”]                     # 숫자형과 문자열을 데이터로 갖는 리스트
e = [1, 2, [“Python”, “C”, “Java”]]      # 숫자형과 리스트를 데이터로 갖는 리스트(중첩 리스트)
l = list()                               # list(함수)
```

➢ 리스트 연산은 더하기와 곱하기 가능

➢ 문자열 연산에서와 동일하게 더하기는 추가, 곱하기는 반복을 의미

```
a = [3, 7, 9] ; b = [4, 6, 10]
print(a + b)              # 리스트 자료형을 가진 변수 a에 변수 b의 데이터를 추가
print(a * 3)              # 리스트 자료형을 가진 변수 a의 데이터들을 세 번 반복
----------------------------------------------------출력결과----------------------------------------------------
[3, 7, 9, 4, 6, 10]
[3, 7, 9, 3, 7, 9, 3, 7, 9]
```

리스트(list)

● 리스트 인덱싱과 슬라이싱

➢ 문자열의 인덱싱과 슬라이싱 표현식과 동일

```
test = [75, 69, 95, 84, 89]
food = ["milk", "juice", "rice", "soup", "cookie"]
print(test[2])                    # 인덱스가 0부터 시작하기 때문에 3번째 자료인 95를 출력
print(food[3:5])                  # 인덱스 3~4에 있는 자료인 'soup', 'cookie'를 출력
----------------------------------------------------출력결과----------------------------------------------------
95
["soup", "cookie"]
```

➢ 중첩된 리스트(리스트 안에 리스트가 있는)의 인덱싱과 슬라이싱

```
a = [1, 2, ["Python", "C", "Java"]]      # 숫자형과 리스트를 데이터로 갖는 리스트(중첩 리스트)
print(a[2][:2])              # 출력된 3번째 데이터(["Python", "C", "Java"])에서 처음부터 인덱스 1까지 슬라이싱
----------------------------------------------------출력결과----------------------------------------------------
["Python", "C"]
```

리스트(list)

● 리스트 변경과 삭제

➢ 변경 표현식 : 변수명[위치] = 변경할 값

```
a = [1, 3, 5, 7, 6, 2, 4]
a[3] = 8                    # 변수 a의 인덱스 3 데이터를 8로 변경
print(a)
```

----------출력결과----------

```
[1, 3, 5, 8, 6, 2, 4]
```

➢ 삭제 표현식 : del 변수명[위치]

```
a = [1, 3, 5, 7, 6, 2, 4]
del a[3]                    # 변수 a의 인덱스 3 데이터 삭제
print(a)
```

----------출력결과----------

```
[1, 3, 5, 6, 2, 4]
```

리스트(list)

● 리스트 함수

➢ append() : 리스트의 마지막 위치에 데이터를 추가

```
a = [1, 2, 3, 4]
a.append(3)                # 변수 a의 마지막 위치에 3을 추가
print(a)
```

----------출력결과----------

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

➢ insert() : 리스트의 원하는 위치에 값 추가

```
a = [1, 2, 3, 4]
a.insert(0, 7)             # 변수 a의 인덱스 0 위치에 데이터 7을 추가
print(a)
```

----------출력결과----------

```
[7, 1, 2, 3, 4]
```

리스트(list)

● 리스트 함수

➢ remove() : 리스트 안의 데이터 삭제

```
a = [1, 2, 3, 4, 5, 4, 3, 2, 1]
a.remove(4)                # remove(x)는 리스트에서 처음으로 일치하는 x 를 삭제
print(a)
```

출력결과

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

➢ 리스트 안의 데이터를 여러 개 삭제해야 한다면 while 반복문을 이용한다.

```
a = [1, 2, 3, 4, 5, 4, 3, 2, 1]
while 4 in a:              # 4 를 발견하면 지워라. 중복 데이터가 많을 때는 시간이 많이 걸린다.
    a.remove(4)
print(a)
```

출력결과

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

리스트(list)

● 리스트 함수

➢ **sort(), reverse() : 리스트 안의 데이터들을 오름차순 또는 내림차순으로 정렬**

```
a = [1, 3, 5, 7, 6, 2, 4]
a.sort() ; print(a)                    # 변수 a의 데이터를 오름차순 정렬
a.reverse() ; print(a)                 # 변수 a의 데이터 순서 뒤집기
----------------------------------------------------출력결과----------------------------------------------------
[1, 2, 3, 4, 5, 6, 7]
[7, 6, 5, 4, 3, 2, 1]
```

➢ **sort(reverse=True) : 내림차순으로 정렬**

```
a = [1, 3, 5, 7, 6, 2, 4]
a.sort(reverse=True)                   # 변수 a의 데이터를 내림차순 정렬
print(a)
----------------------------------------------------출력결과----------------------------------------------------
[7, 6, 5, 4, 3, 2, 1]
```

리스트(list)

● 리스트 함수

➢ pop()

- () 안에 입력이 없으면 리스트 안에서 맨 마지막에 있는 데이터를 보여주고 삭제
- x번 인덱스에 있는 데이터를 보여주고 그 데이터를 삭제

```
a = [1, 3, 5, 7, 6, 2, 4]
print(a.pop())                    # 변수 a의 마지막 데이터를 보여준 다음 삭제
print(a)
```

----------출력결과----------

```
4
[1, 3, 5, 7, 6, 2]
```

```
a = [1, 3, 5, 7, 6, 2, 4]
print(a.pop(3))                   # 변수 a의 인덱스 3 데이터를 보여준 다음 삭제
print(a)
```

----------출력결과----------

```
7
[1, 3, 5, 6, 2, 4]
```

리스트(list)

● 리스트 함수

➢ extend() : 리스트 끝에 다른 리스트의 데이터들을 추가

```
a = [1, 3, 5, 7, 6, 2, 4]
b = [3, 5, 7]
a.extend(b)                    # 변수 a에 변수 b 데이터를 추가함
print(a)
```

출력결과

```
[1, 3, 5, 7, 6, 2, 4, 3, 5, 7]
```

➢ append() 와 extend() 비교

- **append(x)** : 리스트 끝에 x 그 자체를 원소로 넣는다.
- **extend(iterable)** : 리스트 끝에 **iterable**의 모든 항목을 넣는다.

```
a = [1, 3, 5, 7, 6, 2, 4]
b = [3, 5, 7]
a.append(b)                    # 변수 a에 변수 b 데이터를 원소로 추가함
print(a)
```

출력결과

```
[1, 3, 5, 7, 6, 2, 4, [3, 5, 7]]
```

리스트(list)

● 리스트 함수

➢ **in : 리스트에 특정 요소값이 있는지 확인**

```
3 in a
```

----------출력결과----------

```
True
```

➢ **zip() : 2개 리스트의 각 항목을 짝지워 준다.**

- **서로 관련된 두 목록 데이터(list)를 짝지워 주는 용도**
- **짝이 맞지 않는 나머지는 무시됨**

```
z = zip(['a', 'b', 'c'],
        ['d', 'e', 'f', 'g', 'h'])
list(z)
```

----------출력결과----------

```
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

튜플(tuple)

- 튜플(tuple) : ( ) 이용. 리스트와 거의 비슷하나, 데이터 수정 불가
  - 표현식 : 변수명 = (데이터1, 데이터2, 데이터3, ...)

```
tp = tuple()                              # tuple 함수
tp1 = ( )                                 # 괄호 ()
tp2 = (1, )                               # 하나의 데이터만 가진 튜플을 만드는 경우 데이터 뒤에 ,(쉼표) 필수
#tp2 = (1)                                # 정수 1 이 저장됨, tuple이 아니라 정수형 변수임
tp3 = (1, 2, 3)
tp4 = 1, 2, 3                             # ( ) 생략 가능
tp5 = (“a”, “b”, (“ab”, “cd”))
tp6 = tuple("abcd")                       # 반복 가능한 자료형을 튜플 형태로 출력
print(tp); print(tp1); print(tp2); print(tp3); print(tp4); print(tp5); print(tp6)
----------------------------------------------------출력결과----------------------------------------------------
()
()
(1,)
(1, 2, 3)
(1, 2, 3)
('a', 'b', ('ab', 'cd'))
('a', 'b', 'c', 'd')
```

튜플(tuple)

● 튜플(tuple) : ( ) 이용. 리스트와 거의 비슷하나, 데이터 수정 불가

➢ 연산(더하기, 곱하기), 인덱싱과 슬라이싱 가능

```
tp1 = (1, 2, 3, "a", "b", "c")
tp2 = (4, 5, 6)
print(tp1 + tp2)                                  # 튜플 자료형을 가진 변수 tp1과 tb2의 데이터를 합치기
print(tp2 * 2)                                    # 튜플 자료형을 가진 변수 tp2의 데이터를 두 번 반복하기

tp = (1, 2, ("ab", "cd", "ef"), 3, "a", "b", "c")  # 숫자, 문자열, 튜플을 데이터로 갖는 튜플
print(tp[2])                                      # 변수 tp에 저장된 데이터 중 3번째 데이터를 출력
print(tp[2][:2])                                  # 변수 tp의 3번째 데이터에서 처음부터 인덱스 1까지 슬라이싱
----------------------------------------------------출력결과----------------------------------------------------
(1, 2, 3, 'a', 'b', 'c', 4, 5, 6)
(4, 5, 6, 4, 5, 6)

('ab', 'cd', 'ef')
('ab', 'cd')
```

딕셔너리(dictionary)

● 딕셔너리(dictionary) : { } 사전처럼 찾을 단어에 해당하는 key 값, 단어의 의미에 해당하는 value 값으로 구성

➢ 표현식 : 변수명 = { Key1:Value1, Key2:Value2, Key3:Value3, … }

➢ Key는 변하지 않는 값이어야 하고, Value는 변하는 값이나 변하지 않는 값 모두 사용 가능

- Key는 변하지 않는 값이어야 하기 때문에 리스트로 표현할 수 없고, 튜플로 표현할 수 있다.
- Key를 통해 Value를 얻기 때문에 순차 개념이 없어 index() 함수 사용 불가

➢ 딕셔너리 선언

```
dic = {}                                        # 빈 딕셔너리 선언
dic = dict()                                    # dict() 함수 사용. 빈 딕셔너리 선언
dic = {"Key":"Value", "number":1234}            # 여러 데이터 저장 가능
----------------------------------------------------출력결과----------------------------------------------------
{}
{}
{'Key': 'Value', 'number': 1234}
```

딕셔너리(dictionary)

● 딕셔너리 선언 및 출력

➢ 표현식 : 변수명 = { Key1:Value1, Key2:Value2, Key3:Value3, … }

```
dict1 = {"name" : "Lee", "age" : 29, "birth" : "0320"}    # 여러 자료형 입력 예시
dict2 = {1 : "ab"}                                        # Key에 정수, Value에 문자열 자료형 입력한 예
dict3 = {"test" : [1, 3, 5, 7]}                           # Value에 리스트 자료형을 입력한 예
```

➢ Key를 사용해서 Value를 출력

```
dic = {"a" : [1, 2], "b" : [3, 4], "c" : [5, 6]}
print(dic["a"])
----------------------------------------------------출력결과----------------------------------------------------
[1, 2]
```

딕셔너리(dictionary)

● 데이터 추가 및 삭제

➢ 추가 표현식 : 변수명[Key] = Value

```
dic = {“a” : [1, 2]}
dic[“b”] = [3, 4]                    # 변수 dic에 Key:Value가 “b”:[3, 4]인 딕셔너리 쌍을 추가
print(dic)
----------------------------------------------------출력결과----------------------------------------------------
{‘a’ : [1, 2], ‘b’ : [3, 4] }
```

➢ 삭제 표현식 : del 변수명[Key]

```
dic = {“a” : [1, 2], “b” : [3, 4], “c” : [5, 6]}
del dic[“c”]                         # Key “c”에 해당하는 Value [5, 6]을 삭제, Key:Value쌍이 삭제됨
print(dic)
----------------------------------------------------출력결과----------------------------------------------------
{‘a’ : [1, 2], ‘b’ : [3, 4]}
```

딕셔너리(dictionary)

● 딕셔너리 자료형으로 변환

➢ 데이터들이 한 쌍으로 구성된 리스트나 튜플이 있다면 dict() 함수를 이용하여 딕셔너리 자료형으로 변경할 수 있다.

```
list1 = [["a", 2], ["b", 4], ["c", 6]]
dict(list1)                                  # 중첩 리스트로 구성된 자료형을 딕셔너리로 변경
----------------------------------------------------출력결과----------------------------------------------------
{'a' : 2, 'b' : 4, 'c' : 6 }
```

```
list2 = [("a", 2), ("b", 4), ("c", 6)]
dict(list2)                                  # 리스트 내 튜플로 구성된 자료형을 딕셔너리로 변경
----------------------------------------------------출력결과----------------------------------------------------
{'a' : 2, 'b' : 4, 'c' : 6 }
```

```
tp1 = (["a", 2], ["b", 4], ["c", 6])         # 튜플 내 리스트로 구성된 자료형을 딕셔너리로 변경
#tp1 = (("a", 2), ("b", 4), ("c", 6))        # 중첩 튜플로 구성된 자료형을 딕셔너리로 변경
dict(tp1)
----------------------------------------------------출력결과----------------------------------------------------
{'a' : 2, 'b' : 4, 'c' : 6 }
```

딕셔너리(dictionary)

● 자료의 형 변환

```
a = 3                   # 변수 a에 정수형 자료 저장
float(a)                # 변수 a를 실수형으로 변환

a = 3.619               # 변수 a에 실수형 자료 저장
int(a)                  # 변수 a를 정수형으로 변환(소수점 자리는 버림)

b = [1, 2, 3, 4, 5]     # 변수 b에 리스트 자료 저장
tuple(b)                # 변수 b를 튜플형으로 변환

dict(b)                 # 변수 b가 쌍으로 구성되지 않았기 때문에 딕셔너리로 변경 시 에러 발생
```

딕셔너리(dictionary)

● 자료형 확인 : type()

```
a = 5                                          # 정수
b = 3.14                                       # 실수
c = "love"                                     # 문자열
d = True                                       # 불
e = [1, 2, 3, 4, 5]                            # 리스트
f = (6, 7, 8, 9, 10)                           # 튜플
g = {"name":"park","number":"505"}             # 딕셔너리
print(type(a),type(b))
print(type(c),type(d))
print(type(e),type(f),type(g))
```

----------출력결과----------

```
<class 'int'> <class 'float'>
<class 'str'> <class 'bool'>
<class 'list'> <class 'tuple'> <class 'dict'>
```

딕셔너리(dictionary)

● 딕셔너리 함수

➢ keys()와 values() : Key 또는 value만 모아서 리스트 자료형으로 출력

```
dic = {"name" : "Lee", "age" : 29, "birth" : "0320"}
print(dic.keys())                    # 변수 dic의 Key만 모아서 출력
print(list(dic.keys()))              # 변수 dic의 Key만 모인 객체를 리스트로 출력
print(dic.values())                  # 변수 dic의 Value만 모아서 출력
----------------------------------------------------출력결과----------------------------------------------------
dict_keys(['name', 'age', 'birth'])
['name', 'age', 'birth']
dict_values(['Lee', 29, '0320'])
```

➢ items() : Key와 Value 쌍을 튜플 형식으로 출력

```
dic = {"name" : "Lee", "age" : 29, "birth" : "0320"}
dic.items()                  # 변수 dic의 Key와 Value 쌍을 튜플로 출력
----------------------------------------------------출력결과----------------------------------------------------
dict_items([('name', 'Lee'), ('age', 29), ('birth', '0320')])
```

딕셔너리(dictionary)

● 딕셔너리 함수

➢ clear() 함수를 사용해 삭제. 이 때 변수에 들어 있는 값만 지워지고, 변수는 지워지지 않는다.

```
dic = {"name" : "Lee", "age" : 29, "birth" : "0320"}
dic.clear()                                          # 변수 dic 안의 모든 쌍을 삭제
print(dic)
----------------------------------------------------출력결과----------------------------------------------------
{}
```

➢ 딕셔너리 안에 Key 가 있는지 확인하기

```
dic = {"name" : "Lee", "age" : 29, "birth" : "0320"}
print("name" in dic)                    # 변수 dic 안에 "name" Key가 있으면 True, 없으면 False
----------------------------------------------------출력결과----------------------------------------------------
True
```

집합(set)

● 집합(set) : 집합과 관련된 것을 표현하는 자료형

➢ 표현식 : 변수명 = set([데이터1, 데이터2, 데이터3, …]) 또는 변수명 = set(문자열)

➢ 순서가 없어 인덱싱/슬라이싱 불가

➢ 데이터를 모아놓은 것으로 중복을 허용하지 않음

```
s = set()                                  # 비어 있는 집합 자료
set1 = set([1, 2, 5, 3, 7, 4])             # 숫자로 된 리스트의 집합 자료형은 분해되면서 오름차순으로 출력됨
print(s); print(set1)
```

출력결과

```
set()
{1, 2, 3, 4, 5, 7}
```

```
set2 = set("school bus")                   # 문자열의 경우 문자로 분해되고 중복 문자는 제거됨
print(set2)
```

출력결과

```
{'c', 'u', 'l', 'b', 'h', 's', 'o', ' '}
```

집합(set)

● 연산(더하기, 빼기) 가능하며, 함수로 구현 가능

➢ 더하기(|) = 합집합(union) ※ | 은 [shift] + [₩]

➢ 빼기(-) = 차집합(difference)

➢ 공통 데이터(&) = 교집합(intersection)

```
set1 = set([1, 2, 3, 4, 5, 6, 7])
set2 = set([5, 6, 7, 8, 9, 10, 11])
print(set1.union(set2))                 # print(set1 | set2)와 결과는 동일
print(set1.difference(set2))            # print(set1 - set2)와 결과는 동일
print(set2.difference(set1))            # print(set2 – set1)와 결과는 동일
print(set1.intersection(set2))          # print(set1 & set2)와 결과는 동일)
----------------------------------------------------출력결과----------------------------------------------------
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
{1, 2, 3, 4}
{8, 9, 10, 11}
{5, 6, 7}
```

집합(set)

● 집합 자료형에 add(), update(), remove() 함수 이용

➢ 데이터 한 개 추가

```
set1 = set([1, 2, 3, 5])
set1.add(4)
set1
```

----------출력결과----------

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

➢ 여러 개의 데이터 추가

```
set2 = set([1, 2, 3, 4, 5])
set2.update([6, 7, 8])
set2
```

----------출력결과----------

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

집합(set)

● 집합 자료형에 add(), update(), remove() 함수 이용

➢ 특정 데이터 삭제

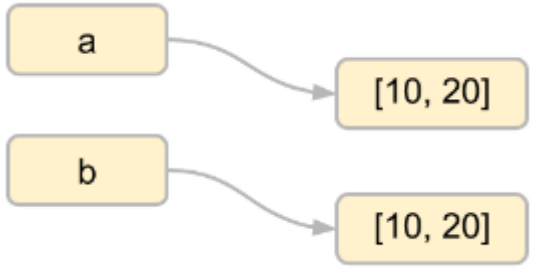
```
set3 = set([1, 2, 3, 4, 5])
set3.remove(3)
set3
```

----------출력결과----------

```
{1, 2, 4, 5}
```

객체와 할당

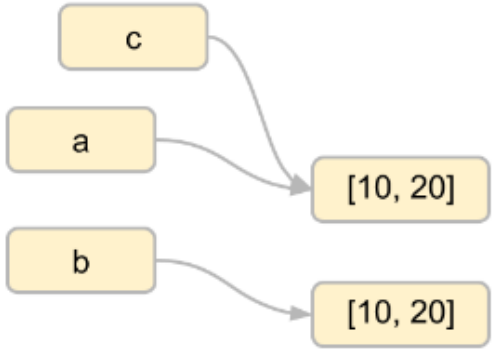
- 파이썬에서는 모든 것이 객체다.
- a와 b가 같은 값을 가지고 있지만 실제로 메모리(기억장소)에서는 다른 객체다.
- [10, 20] 자체가 하나의 객체이며, 변수 a와 b는 이 객체를 "**가리키고 있다**"고 표현하는 것이 더 정확한 표현이다.
- 즉, 변수 a와 b는 데이터를 저장한 객체가 아니라 데이터 객체에 대한 **참조**이다.




```
a = [10, 20]
b = [10, 20]
print( a == b )          # 값을 비교
print( a is b )          # 같은 객체인지 비교
```

----------출력결과----------

```
True
False
```




```
1 print(id(a))
2 print(id(b))
3 print(id(c))
```

```
2223631829312
2223624685120
2223631829312
```

```
c = a
print( a == c )          # 값을 비교
print( a is c )          # 같은 객체인지 비교
```

----------출력결과----------

```
True
True
```