

자료형과 제어문

Python 튜터링 튜터
소프트웨어학부 손민우

02장 파이썬 프로그래밍의 기초, 자료형

어떤 프로그래밍 언어든 "그 언어의 자료형을 알고 이해할 수 있다면 이미 그 언어의 절반을 터득한 것이나 다름없다"는 말이 있다.

자료형이란 프로그래밍을 할 때 쓰이는 숫자, 문자열 등 자료 형태로 사용하는 모든 것을 뜻한다. 프로그램의 기본이자 핵심 단위가 바로 자료형이다. 계산 프로그램을 만들려면 어떤 것을 계산할지부터 알아야 하고, 데이터베이스 프로그램을 만들려면 어떤 자료를 저장할지부터 알아야 하는 것처럼 기본 중의 기본이다. 따라서 자료형을 충분히 이해하지 않고 프로그래밍을 시작하려는 것은 기초 공사가 마무리되지 않은 상태에서 빌딩을 세우는 것과 같다.

자료형에는 어떤 것이 있는지 이 장에서 하나씩 자세하게 알아보자.

목차

1. 변수란?
2. 숫자형
3. 문자열
4. 리스트
5. 딕셔너리
6. 튜플과 집합
7. 불 자료형

8. If문
9. while문
10. for문

변수

변수 이름 = 변수에 저장할 값

`a = [1, 2, 3]`

만약 위 코드처럼 `a = [1, 2, 3]` 이라고 하면 `[1, 2, 3]` 값을 가지는 리스트 자료형(객체)이 자동으로 메모리에 생성되고 변수 `a` 는 `[1, 2, 3]` 리스트가 저장된 메모리의 주소를 가리키게 된다.

※ 메모리란 컴퓨터가 프로그램에서 사용하는 데이터를 기억하는 공간이다.

```
a = [1, 2, 3]  
id(a)
```

~~1773505820424~~

```
b = a  
id(b)
```

1773505820424

수정 가능한가?

여러 자료를 담는가?

담겨져 있는 자료들을
변경할 수 있는가?

숫자형

문자

리스트, 딕셔너리, 불

튜플, 집합, 문자열

숫자형

정수형

그 외 수



정수형(Integer)이란 말 그대로 정수를 뜻하는 자료형을 말한다. 다음 예는 양의 정수와 음의 정수, 숫자 0을 변수 a에 대입하는 예이다.

```
>>> a = 123
>>> a = -178
>>> a = 0
```

- ~ 0 ~ +

실수형

파이썬에서 실수형(Floating-point)은 소수점이 포함된 숫자를 말한다. 다음은 실수를 변수 a에 대입하는 예이다.

```
>>> a = 1.2
>>> a = -3.45
```

위 방식은 우리가 일반적으로 볼 수 있는 실수형의 소수점 표현 방식이다.

```
>>> a = 4.24E10
>>> a = 4.24e-10
```

위 방식은 "컴퓨터식 지수 표현 방식"으로 파이썬에서는 4.24e10 또는 4.24E10처럼 표현한다(e와 E 둘 중 어느 것을 사용해도 무방하다). 여기서 4.24E10은 $4.24 * 10^{10}$, 4.24e-10은 $4.24 * 10^{-10}$ 을 의미한다.

숫자형

사칙연산

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
>>> a / b
0.75
```

Handwritten notes: $a + b$ has a red underline under the plus sign. $a * b$ has a red underline under the asterisk and a red 'X' next to it. a / b has a red underline under the slash and a red arrow pointing to the result 0.75.

제곱

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

Handwritten notes: $a ** b$ has a red underline under the double asterisk. The result 81 has a red underline. There are red handwritten numbers 3 and 4 next to the result.

나머지와 몫

```
>>> 7 % 3
1
>>> 3 % 7
3
```

Handwritten notes: The first example has a red arrow pointing to the result 1. The second example has a red arrow pointing to the result 3.

```
>>> 7 // 4
1
```

Handwritten notes: The result 1 is circled in red. There is a red arrow pointing to the result.

기타 수학 연산들

Import math
라이브러리 사용!

반올림, 올림, 내림

순열, 조합

상수 e, π

제곱근 등등..

문자열

1. 큰따옴표(")로 양쪽 둘러싸기

```
"Hello World"
```

2. 작은따옴표(')로 양쪽 둘러싸기

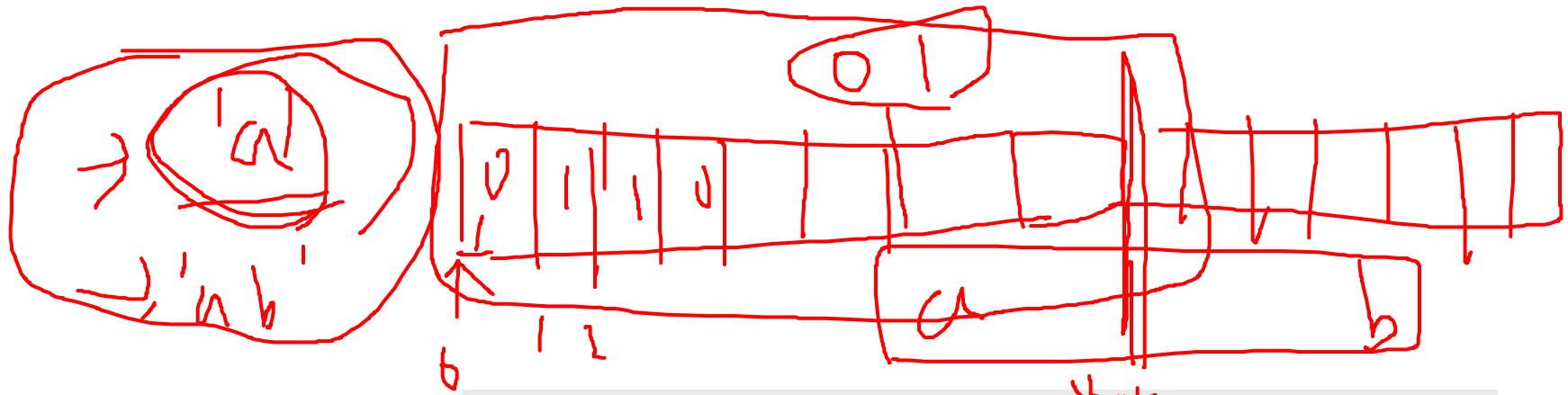
```
'Python is fun'
```

3. 큰따옴표 3개를 연속(""")으로 써서 양쪽 둘러싸기

```
"""Life is too short, You need python"""
```

4. 작은따옴표 3개를 연속(''')으로 써서 양쪽 둘러싸기

```
'''Life is too short, You need python'''
```



```
multiline = "Life is too short\nYou need python"
```

`\t`

문자열 사이에 탭 간격을 줄 때 사용

`\\`

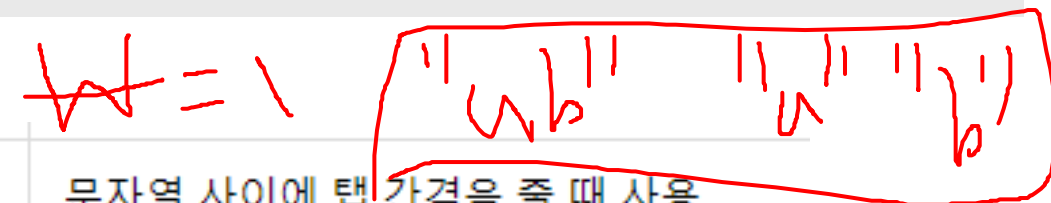
문자 `\`를 그대로 표현할 때 사용

`\'`

작은따옴표(')를 그대로 표현할 때 사용

`\"`

큰따옴표(")를 그대로 표현할 때 사용



문자열

문자열 더해서 연결하기(Concatenation)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

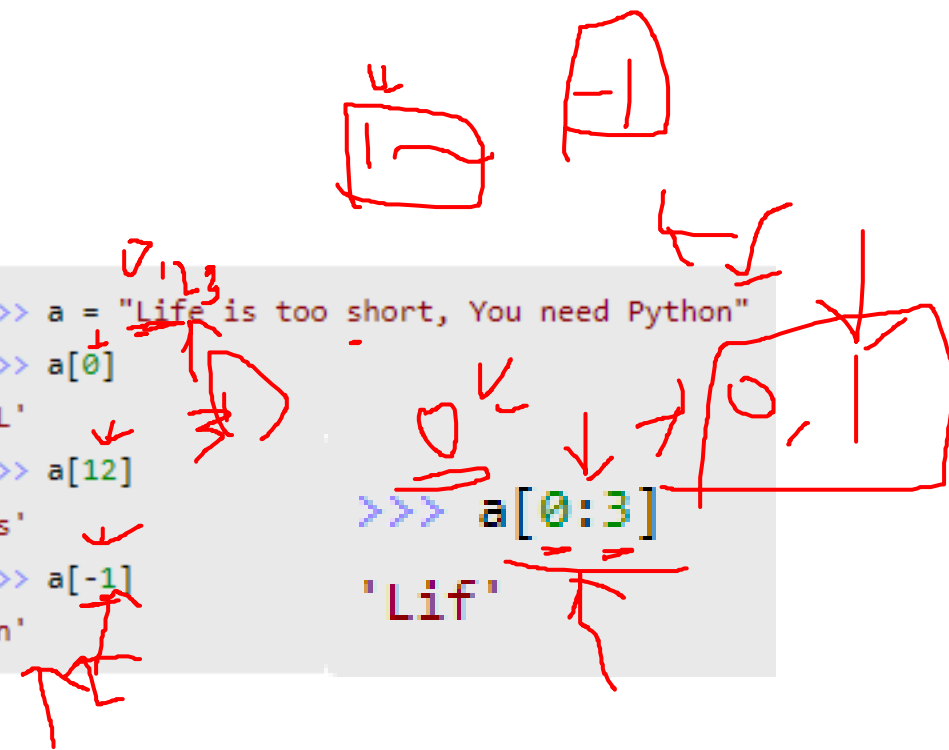
문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

문자열 길이 구하기

문자열의 길이는 다음과 같이 len 함수를 사용하면 구할 수 있다.
정 없이 바로 사용할 수 있다.

```
>>> a = "Life is too short"
>>> len(a)
17
```



```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
>>> a[0:3]
'Lif'
```


문자열

~~4-1(X)~~ ←

hi 8

format()

다음과 같이 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능을 사용할 수 있다.

```
>>> name = '홍길동'
>>> age = 30
>>> # '나의 이름은 {name}입니다. 나이는 {age}입니다.'
>>> '나의 이름은 홍길동입니다. 나이는 30입니다.'
```

```
>>> age = 30
>>> f'나는 내년이면 {age+1}살이 된다.'
>>> '나는 내년이면 31살이 된다.'
```

정렬은 다음과 같이 할 수 있다.

```
>>> f'{"hi":<10}' # 왼쪽 정렬
'hi'
>>> f'{"hi":>10}' # 오른쪽 정렬
'      hi'
>>> f'{"hi":^10}' # 가운데 정렬
'      hi'
```

공백 채우기는 다음과 같이 할 수 있다.

```
>>> f'{"hi":^10}' # 가운데 정렬하고 '=' 문자로 공백 채우기
'====hi===='
>>> f'{"hi":!<10}' # 왼쪽 정렬하고 '!' 문자로 공백 채우기
'hi!!!!!!!!'
```

리스트

리스트를 사용하면 1, 3, 5, 7, 9 숫자 모음을 다음과 같이 간단하게 표현할 수 있다.

```
>>> odd = [1, 3, 5, 7, 9]
```

리스트를 만들 때는 위에서 보는 것과 같이 대괄호([])로 감싸 주고 각 요소값은 쉼표(,)로 구분해 준다.

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

여러 가지 리스트의 생김새를 살펴보면 다음과 같다.

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

```
>>> a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
>>> a[2:5]
[3, ['a', 'b', 'c'], 4]
>>> a[3][:2]
['a', 'b']
```

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

다음 예를 따라 해 보자.

```
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
```

```
>>> a[-1][1]
'b'
>>> a[-1][2]
'c'
```

리스트

리스트 더하기(+)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

리스트 반복하기(*)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트 요소 끄집어내기(pop)

pop()은 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제한다.

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```

리스트 길이구하기

리스트 길이를 구하기 위해서는 다음처럼 len 함수.

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트에서 값 수정하기

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

del 함수 사용해 리스트 요소 삭제하기

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

리스트에 요소 추가(append)

append를 사전에서 검색해 보면 "덧붙이다, 첨부하다"라는 뜻이 있다.
는 리스트의 맨 마지막에 x를 추가하는 함수이다.

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

딕셔너리

딕셔너리는 어떻게 만들까?

다음은 기본 딕셔너리의 모습이다.

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

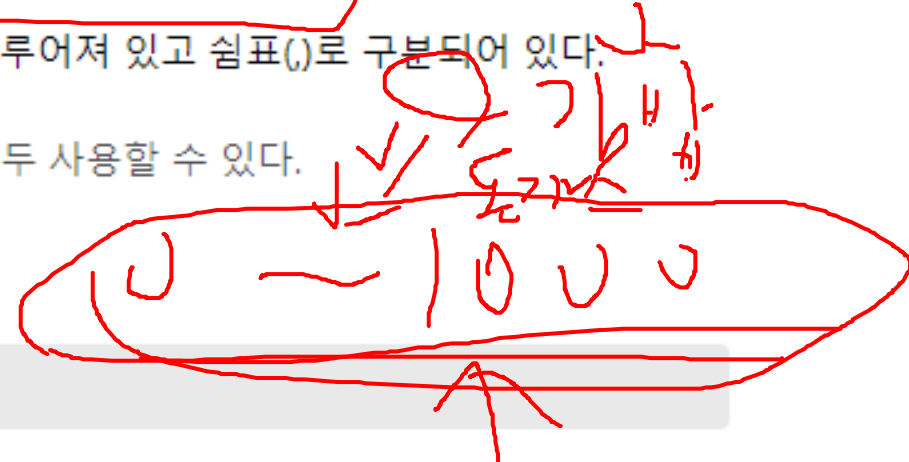
Key와 Value의 쌍 여러 개가 {}로 둘러싸여 있다. 각각의 요소는 Key : Value 형태로 이루어져 있고 쉼표(,)로 구분되어 있다.

※ Key에는 변하지 않는 값을 사용하고, Value에는 변하는 값과 변하지 않는 값 모두 사용할 수 있다.

다음 딕셔너리 예를 살펴보자.

```
>>> dic = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
```

위에서 Key는 각각 'name', 'phone', 'birth'이고, 각각의 Key에 해당하는 Value는 'pey', '0119993323', '1118'이 된다.



딕셔너리

딕셔너리 쌍 추가하기

```
>>> a = {1: 'a'}  
>>> a[2] = 'b'  
>>> a  
{1: 'a', 2: 'b'}
```

딕셔너리 요소 삭제하기

```
>>> del a[1]  
>>> a  
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

딕셔너리에서 Key 사용해 Value 얻기

다음 예를 살펴보자.

```
>>> grade = {'pey': 10, 'julliet': 99}  
>>> grade['pey']  
10  
>>> grade['julliet']  
99
```

해당 Key가 딕셔너리 안에 있는지 조사하기(in)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}  
>>> 'name' in a  
True  
>>> 'email' in a  
False
```

튜플과 집합

튜플의 모습은 다음과 같다.

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

집합 자료형은 다음과 같이 set 키워드를 사용해 만들 수 있다.

```
>>> s1 = set([1, 2, 3])
>>> s1
{1, 2, 3}
```

2. 합집합

합집합은 다음과 같이 구할 수 있다.

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

3. 차집합

차집합은 다음과 같이 구할 수 있다.

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```

1. 교집합

s1과 s2의 교집합을 구해 보자.

```
>>> s1 & s2
{4, 5, 6}
```

불 자료형

불 자료형이란?

불(bool) 자료형이란 참(True)과 거짓(False)을 나타내는 자료형이다. 불 자료형은 다음 2가지 값만을 가질 수 있다.

- True - 참
- False - 거짓

※ True나 False는 파이썬의 예약어로 true, false와 같이 사용하지 말고 첫 문자를 항상 대문자로 사용해야 한다

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

```
>>> 2 > 1
True
```

```
>>> bool('python')
True
```

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
0	거짓
{}	거짓
1	참
0	거짓
None	거짓

If문

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else:
    수행할 문장A
    수행할 문장B
    ...
```

```
>>> money = True
>>> if money:
...     print("택시를 타고 가라")
... else:
...     print("걸어 가라")
...
택시를 타고 가라
```

비교연산자

비교연산자	설명
$x < y$	x가 y보다 작다
$x > y$	x가 y보다 크다
$x == y$	x와 y가 같다
$x != y$	x와 y가 같지 않다
$x >= y$	x가 y보다 크거나 같다
$x <= y$	x가 y보다 작거나 같다

연산자	설명
$x \text{ or } y$	x와 y 둘중에 하나만 참이어도 참이다
$x \text{ and } y$	x와 y 모두 참이어야 참이다
$\text{not } x$	x가 거짓이면 참이다

in

x in 리스트

not in

x not in 리스트

x in 튜플

x not in 튜플

x in 문자열

x not in 문자열

즉 elif는 이전 조건문이 거짓일 때 수행된다.

```
If <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
else:
    <수행할 문장1>
    <수행할 문장2>
    ...
```


while문

while문의 기본 구조

반복해서 문장을 수행해야 할 경우 while문을 사용한다. 그래서 while문을 반복문이라고도 부른다.

다음은 while문의 기본 구조이다.

```
while <조건문>:
```

```
    <수행할 문장1>
```

```
    <수행할 문장2>
```

```
    <수행할 문장3>
```

```
    ...
```

```
>>> coffee = 10
```

```
>>> money = 100
```

```
>>> while money:
```

```
...     print("돈을 받았으니 커피를 줍니다.")
```

```
...     coffee = coffee - 1
```

```
...     print("남은 커피의 양은 %d개입니다." % coffee)
```

```
...     if coffee == 0:
```

```
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
```

```
...         break
```

```
... 
```

```
>>> a = 0
>>> while a < 10:
...     a = a + 1
...     if a % 2 != 0: continue
...     print(a)
```

```
1
3
5
7
9
```

파이썬에서 무한 루프는 while문으로 구현할 수 있다.

```
while True:
```

```
    수행할 문장1
```

```
    수행할 문장2
```

```
    ...
```

for문

for문의 기본 구조

for문의 기본 구조는 다음과 같다.

```
for 변수 in 리스트(또는 튜플, 문자열):  
    수행할 문장1  
    수행할 문장2  
    ...
```

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 "수행할 문장1", "수행할 문장2" 등이 수행된다.

```
>>> test_list = ['one', 'two', 'three']  
>>> for i in test_list:  
...     print(i)  
...  
one  
two  
three
```

```
>>> add = 0  
>>> for i in range(1, 11):  
...     add = add + i  
...  
>>> print(add)  
55
```

[1, 2, ..., 10]

Q & A