



## 제3장 자료형(Data-Type)

# 1. 자료형(data-type)

## 1) 파이썬의 자료형

- 파이썬이 처리하는 자료형에는 3가지 종류가 있다.

① 정수(integer) : 1,2,3,4 ...

② 실수(floating-point) : 1.1, 1.2, 1.3 ...

③ 문자열(string) : "hi", "hello" ...

자료형	예
정수	..., -2, -1, 0, 1, 2, ...
실수	3.2, 3.14, 0.12
문자열	'Hello World!', "123"

- type()함수는 자료형을 알고 싶을 때 사용한다.

```
>>> type("Hello World!")
<class 'str'>
>>> type(3.2)
<class 'float'>
>>> type(17)
<class 'int'>
```

# 1. 자료형(data-type)

## 1) 파이썬의 자료형

- 파이썬의 변수에는 어떤 자료형의 데이터든지 저장 가능하다. 또 중간에 다른 자료형의 데이터를 저장하여도 된다.

```
>>> x = 3.2
>>> x = "hello"
>>> x
hello
```

- 하여, 일상적인 경우에 개발자는 데이터의 종류에 신경 쓰지 않아도 된다. 하지만 문자열과 숫자는 서로 구분하여야 한다. 문자열 "10"과 숫자 10은 CPU에서 아주 다르게 취급된다. 많이 사용하는 함수 중 input()함수가 있다. input()함수는 사용자로부터 텍스트 형태의 데이터를 받아서 반환하는 함수이다.

```
>>> x = input("정수 입력 : ")
정수 입력 : 55
>>> x
"55"
```

- 위와 같이 input()함수가 반환한 것은 문자열 "55"인 것을 알 수 있다. 이 때 정수로 바꾸고 싶다면 int()함수를 사용하도록 해야 한다. 아울러 문자열을 실수로 변경해주는 함수는 float()이다.

# 1. 자료형(data-type)

## 1) 파이썬의 자료형

- 이번에는 반대로 정수를 문자열로 변환하고자 한다면 `str()`함수를 사용하면 된다.

```
>>> print("나는 " + str(3) + "학년이다")  
나는 3학년이다.
```

- 하여, 파이썬에서는 정수 10과 "10"을 비교하면 같지 않다고 나온다. 하지만 정수 10과 실수 10.0을 비교하면 같다고 출력된다. 파이썬에서 값이 같은지를 비교하는 연산자는 비교연산자인 `==`을 사용한다.

```
>>> 10 == "10"  
False  
>>> 10 == 10.0  
True
```

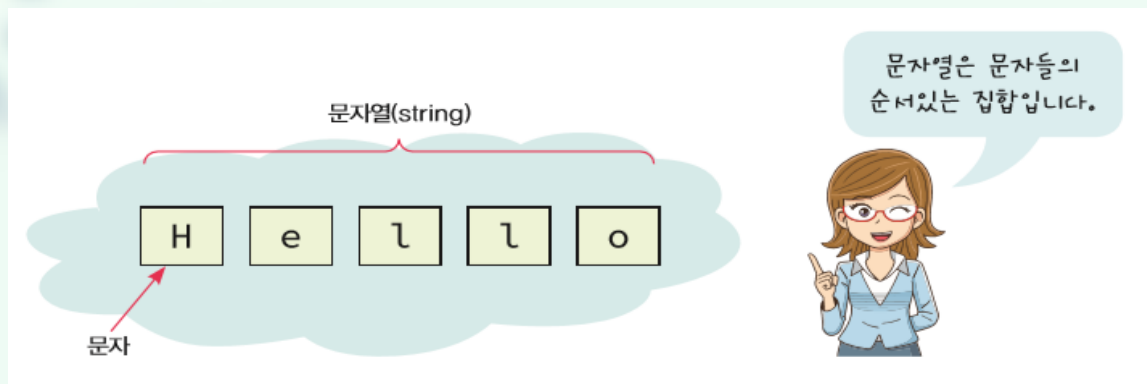
## 2. 문자열(string)

### 1) 파이썬의 문자열

- CPU는 0,1로 연산을 하므로 숫자가 중요하지만 인간에게는 문자열(string)을 사용하여 정보를 표현하고 저장하므로 문자열의 처리도 매우 중요한 부분이다. 프로그래밍을 할 때 아주 많이 접하는 문제가 문자열을 나누고 합치고 문자열에서 특정한 단어를 검색하는 작업이 있을 것이다.

예) 이메일 주소 ["aaa@google.com"](mailto:aaa@google.com)에서 '@'문자를 중심으로 아이디와 도메인을 분리하는 문제를 생각해보면 타 프로그래밍 언어에서는 문자열을 처리하는 작업이 꽤 복잡하다. 하지만, 파이썬에서는 최근에 개발된 언어이니만큼 문자열 처리하는 작업이 놀랄 정도로 간단하고 직관적이다.

- 문자열(string)은 문자들의 순서있는 집합(sequence of characters)이다. 프로그래머가 아닌 사람들은 이것을 텍스트 데이터라고 부른다. 프로그래머들만 텍스트 데이터를 문자열이라고 부른다. 전문적인 용어를 사용해야만 혼돈이 없기 때문이다. 문자열은 아래 그림처럼 일렬로 나열된 문자들이다.



## 2. 문자열(string)

### 1) 파이썬의 문자열

- 파이썬에서는 큰따옴표("...")로 텍스트를 감싸면 문자열이 된다.
- 문자열은 변수에 저장될 수 있다. 변수에 저장된 문자열을 print()함수를 이용하여 출력한다.  
아니면 변수이름만 입력하고 엔터키를 눌러도 출력된다.
- 아울러, 작은따옴표('...')를 사용해도 문자열을 만들 수 있다. 하지만 문자열을 큰따옴표(")로 시작했다가 작은따옴표(')로 끝내면 문법적인 오류가 생긴다.(주의)
- 아울러 따옴표로 시작했는데 단어의 끝에 따옴표가 없어도 문법적인 오류가 된다.

```
>>> greeting="Hello'  
SyntaxError: EOL while scanning string literal  
>>>  
>>> greeting=" Hello  
SyntaxError: EOL while scanning string literal
```

- 문법(syntax)이라는 것은 컴퓨터에서는 프로그램의 문장을 바르게 구성하기 위한 규칙을 의미한다.
- 위에서 EOL이라는 것은 “End Of Line” 즉 줄의 끝을 만났다는 의미이다. 큰 따옴표가 있을 것으로 기대하였는데 줄의 끝을 만날 때까지 발견하지 못했다는 의미가 되는 것이다.

## 2. 문자열(string)

### 1) 파이썬의 문자열

- 파이썬에서는 왜 문자열을 나타내는데 큰따옴표와 작은따옴표를 동시에 사용하는 이유가 무엇일까? 그것은 바로 문자열 안에 따옴표가 들어가는 경우를 처리하기 위해서이다. 아래 문장을 보도록 하자.

```
>>> message="철수가 "안녕"이라고 말했습니다." # 컴파일러가 혼돈을 일으킨다.  
SyntaxError: invalid syntax  
>>> message="철수가 '안녕'이라고 말했습니다."  
>>> print(message)  
철수가 '안녕'이라고 말했습니다.  
>>>
```

- 파이썬에서는 따옴표를 출력할 때 \를 사용할 수 있는데, 문자 앞에 \가 붙으면 문자의 특수한 의미를 잃어버린다. 따옴표 앞에 붙이면 문자열을 나타내는 따옴표의 특수한 의미를 잃어버리고 하나의 문자가 된다.

```
>>> message= 'doesn\'t' # \를 사용하여 작은따옴표를 출력한다.  
>>> print(message)  
doesn't  
>>> message="\"Yes,\" he said."  
>>> print(message)  
"Yes," he said.
```

## 2. 문자열(string)

### 1) 파이썬의 문자열

- 아울러 \n은 줄바꿈 문자를 나타내는 특수한 문자이다. 문자열의 중간에 \n이 있으면 줄바꿈을 해서 출력한다.

```
>>> a = "첫 번째 줄\n두 번째 줄"
>>> print(a)
첫 번째 줄
두 번째 줄
```

- 또한, print()함수 사용시에 \가 앞에 붙은 문자를 특수 문자로 취급하고 싶지 않다면 첫 번째 따옴표 앞에 r을 추가하여 특수 문자의 의미를 없앨 수 있다.

```
>>> print("c:\temp\name") # 여기서는 \n은 줄바꿈으로 해석된다.
c:\temp
ame
>>> print(r"c:\temp\name") #문자열 앞에 r을 붙이면 특수문자로 해석하지 않는다
c:\temp\name
```

- 문자열의 길이는 len()함수를 사용하면 알 수가 있다.

```
>>> len("hello")
5
```



## 2. 문자열(string)

### 1) 파이썬의 문자열

- 이스케이프 문자란 일반 문자가 아니고 시스템을 제어하기 위한 특수한 문자이다. 파이썬에서 사용되는 이스케이프 문자에는 아래와 같다.

이스케이프 문자	출력되는 문자
\\	백슬래시(\)
\'	작은 따옴표(')
\"	큰 따옴표(“)
\n	줄 바꿈 문자
\t	탭 문자

## 2. 문자열(string)

### 2) 문자열의 연결

- 파이썬 셸에서 2개 이상의 문자열 리터럴(즉 따옴표로 감싸진 문자열)이 서로 붙어 있으면 자동으로 연결된다.  
이것은 리터럴에만 허용된다.

```
>>> 'Py' 'thon'  
'Python'
```

- 변수나 수식은 해당하지 않지만 만약 변수와 변수, 또는 변수와 리터럴을 연결하고 싶으면 + 연산자로 합칠 수 있다.  
아래와 같이 되는 것을 문자열 접합이라고 한다.

```
>>> 'Py' + 'thon'  
'Python'  
  
>>> first_name="길동"  
>>> last_name="홍"  
>>> name = last_name + first_name  
>>> print(name)  
홍길동
```

## 2. 문자열(string)

### 3) 문자열과 정수 간의 변환

- + 연산자는 2개의 문자열을 합치거나 2개의 정수를 합칠 수 있지만 만약 아래와 같은 경우 즉, 문자열과 정수를 합치라고 한다면 오류가 발생한다.

```
>>> "Student"+26
...
TypeError: Can't convert 'int' object to str implicitly
```

- 그 이유는 바로 파이썬에서는 모든 데이터에는 타입(type)이 있다. 자료형이라고도 한다. 위에서 “Student”는 문자열 타입이고 26은 정수 타입이다. 타입이 다른 데이터를 + 연산자로 합치려고 시도하면 오류가 발생하는 것이다. 26을 str()을 이용하여 문자열로 변환한 후에 합쳐야 하는 것이다.

```
>>> "Student"+str(26)
'Student26'
```

- 반대로 문자열을 숫자로 변환하는 함수도 있다.아래와 같이 말이다.

```
>>> price = int("259000")      # 문자열을 정수로
>>> height = float(" 290.54 "); # 문자열을 실수로
```

자료형(Data Type)은 프로그래밍 언어에서 정수값, 실수값 등의 여러 종류의 데이터를 식별하는 분류로써 데이터 타입 또는 줄여서 타입이라고도 한다. 파이썬은 변수를 생성할 때, 자료형을 적지 않지만 C언어, 자바 등은 다른 언어에서는 변수를 생성할 때 변수의 자료형을 명시하여야 한다.

## 2. 문자열(string)

### 4) 문자열의 반복

- 파이썬에서 특이한 점은 동일한 문자열을 반복시켜서 새로운 문자열을 생성할 수 있다는 점이다. 예를 들어서 "="을 반복하여 "======"과 같은 줄을 손쉽게 만들 수 있다.

```
>>> line = "=" * 50
>>> print(line)
=====
```

- 어떠한 문자열도 \* 연산자를 이용하여서 반복시킬 수 있다. 예를 들어서 "Congratulation!" 를 3번 되풀이 하려면 아래와 같이 작성하면 된다.

```
>>> message = "Congratulations! "
>>> print(message*3)
Congratulations! Congratulations! Congratulations!
```

## 2. 문자열(string)

### 5) 문자열의 출력

- 문자열에 변수의 값을 삽입하여 출력하고 싶으면 **%s** 를 이용한다. 예를 들어서 물건의 가격을 변수에 저장한 후에 “상품의 가격은 10000원입니다.”와 같이 출력한다고 하면 아래와 같이 하면 된다.

```
>>> price = 10000
>>> print("상품의 가격은 %s원입니다." % price)
상품의 가격은 10000원입니다.
```

위의 코드에서 10000을 가지는 변수 price가 생성되었다. print()함수에서 상품의 가격이 들어갈 부분은 %s로 표시되었다. %s 자리에 price값의 값을 출력하라고 알려주는 것이다. 물론 처음부터 “상품의 가격은 10000원입니다.”라고 하여도 되지만 상품의 가격은 항상 변할 수 있기에 변수를 사용하는 것이 좋다.

```
>>> message = "현재 시간은 %s입니다."
>>> time = "12:00pm"
>>> print(message % time)
현재 시간은 12:00pm입니다.
```

문자열 안에서 하나 이상의 %s 를 사용할 수도 있다. 이때는 값들을 괄호로 묶어서 % 뒤에 명기해야 한다.

```
>>> message = "오늘은 %s월 %s일입니다."
>>> print(message % (3, 1))
오늘은 3월 1일입니다.
```

## 2. 문자열(string)


### 6) 인덱싱

- 문자열 중에서 하나의 문자를 추출하려면 어떻게 해야 할까? 예를 들어서 암호화 프로그램에서는 문자열에서 하나의 문자를 추출하는 것이 필요하다. **인덱싱(Indexing)이란 문자열에 [와 ]을 붙여서 문자를 추출하는 것이다.**

[와 ] 사이에는 인덱스라는 숫자가 들어간다. 인덱스(index)는 문자열에 포함된 각각의 문자에 매겨진 번호이다.

예를 들어서 문자열 “python”에서 각 문자의 인덱스는 아래와 같다.

P	y	t	h	o	n
0	1	2	3	4	5



인덱스는 문자에 매겨진 번호입니다. 0부터 시작해요!

- 첫 번째 문자의 인덱스는 0이다. 두 번째 문자는 1이고 세 번째 문자는 2가 된다. 인덱스에서 헤깔리는 것이 바로 0부터 시작한다는 점이다.
- 아울러 인덱스는 음수가 될 수도 있다. 이것은 파이썬의 특별한 기능이다. 인덱스가 음수가 되면 오른쪽에서 왼쪽으로 번호가 매겨진다.

P	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

음수 인덱스는 -1부터 시작한다는 것을 알도록 하자

## 2. 문자열(string)

### 6) 인덱싱

```
>>> word = 'Python'
>>> word[0]
'P'
>>> word[5]
'n'
```

- 위와 같이 인덱스 0은 'P'라는 문자이며 인덱스 5는 'n'이다. 하여 n-1로 생각하면 편하다.

```
>>> word = 'Python'
>>> word[50]
IndexError : string index out of range
```

- 인덱스에 해당하지 않는 수를 입력하면 위와 같이 인덱스의 범위를 벗어났다는 에러가 출력된다.

```
>>> word = 'Python'
>>> word[0] = 'C'
TypeError : str object does not support item assignment
```

- 파이썬에서 한번 작성된 문자열은 변경이 불가능하다. 따라서 위와 같이 0번째 인덱스에 문자를 바꿀려고 하면 에러가 발생하는 것이다.

# 3. 리스트

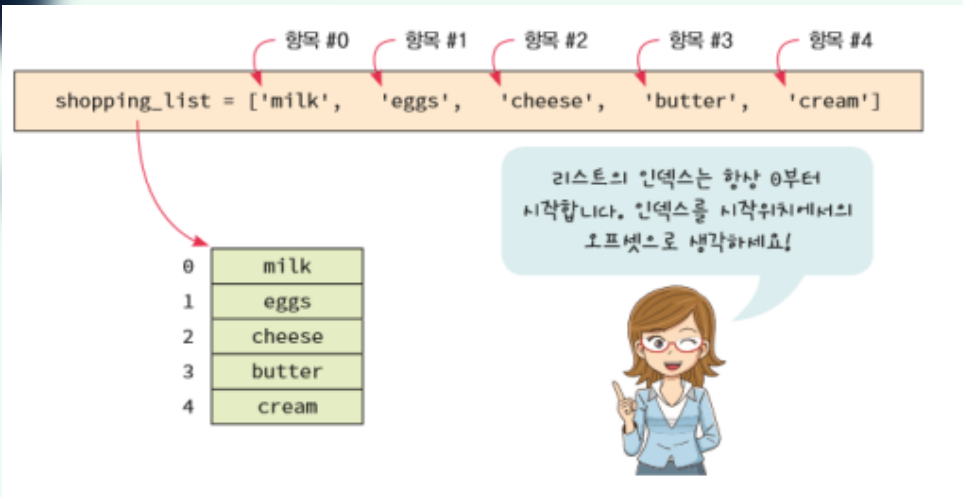
## 1) 리스트

- 파이썬은 여러 개의 값을 모아서 하나의 변수에 저장할 수 있다. 가장 널리 사용되는 것은 리스트(list)이다. 리스트는 목록 또는 일람표라고 할 수 있다. 리스트는 [ ] 안에 값을 나열하고 값과 값 사이에 콤마(,)를 찍으면 된다.

예를 들면 아래와 같다.

```
>>> shopping_list = ['milk', 'eggs', 'cheese', 'butter', 'cream']
>>> print(shopping_list)
['milk', 'eggs', 'cheese', 'butter', 'cream']
>>>
```

- 파이썬에서 리스트는 아주 유용하다. 그 이유는 필요에 따라서 리스트를 조작할 수 있기 때문이다. 즉, 리스트의 항목을 삭제하거나 교체할 수 있다는 의미이다. 먼저 파이썬의 리스트 안에 저장된 항목들은 번호를 가지고 있다.





## 3. 리스트

### 1) 리스트

- 하여, 리스트의 내용을 번호(인덱스)를 가지고 특정한 항목을 출력할 수가 있다.

```
>>> print(shopping_list[2])  
cheese
```

- 아울러, 아래와 같이 항목 변경도 가능하다. 리스트는 변경가능한 객체이다.

```
>>> shopping_list[2]='apple'  
>>> print(shopping_list)  
['milk', 'eggs', 'apple', 'butter', 'cream']  
>>>
```

# 3. 파이썬 튜터

## 1) 파이썬 튜터

- 어떠한 소스코드를 보다가 모르는 부분이 생기면 타 언어들도 마찬가지로 documents가 존재한다.  
즉 documents는 소스코드를 설명해 주는 문서이다.
- <http://www.pythontutor.com> 에 접속하여서 코드를 입력하고 “Visualize Execution”버튼을 누르면 된다.

The screenshot displays the Python Tutor interface for Python 3.6. On the left, a code editor shows three lines of Python code: `shopping_list = ['milk', 'eggs', 'cheese', 'butter', 'cream']`, `print(shopping_list)`, and `['milk', 'eggs', 'cheese', 'butter', 'cream']`. The first line is highlighted with a green arrow, indicating it was just executed. The second line is highlighted with a red arrow, indicating it is the next line to execute. Below the code editor, there are navigation buttons: `<< First`, `< Prev`, `Next >`, and `Last >>`. A progress bar shows the current step is 3 of 3. On the right, the 'Print output' section shows the output of the code: `['milk', 'eggs', 'cheese', 'butter', 'cream']`. Below this, the 'Frames' and 'Objects' sections are shown. The 'Frames' section shows the 'Global frame' with the variable 'shopping\_list'. The 'Objects' section shows a list object 'list' with five elements: 'milk', 'eggs', 'cheese', 'butter', and 'cream'. An arrow points from the 'shopping\_list' variable in the 'Global frame' to the 'list' object in the 'Objects' section.

Python 3.6  
([known limitations](#))

```
1 shopping_list = ['milk', 'eggs', 'cheese', 'butter', 'cream']  
→ 2 print(shopping_list)  
→ 3 ['milk', 'eggs', 'cheese', 'butter', 'cream']
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

`<< First` `< Prev` `Next >` `Last >>`

Step 3 of 3

[Customize visualization](#) (NEW!)

Print output (drag lower right corner to resize)

```
['milk', 'eggs', 'cheese', 'butter', 'cream']
```

Frames

Global frame

shopping\_list

Objects

list

0	1	2	3	4
"milk"	"eggs"	"cheese"	"butter"	"cream"



**감사합니다.**