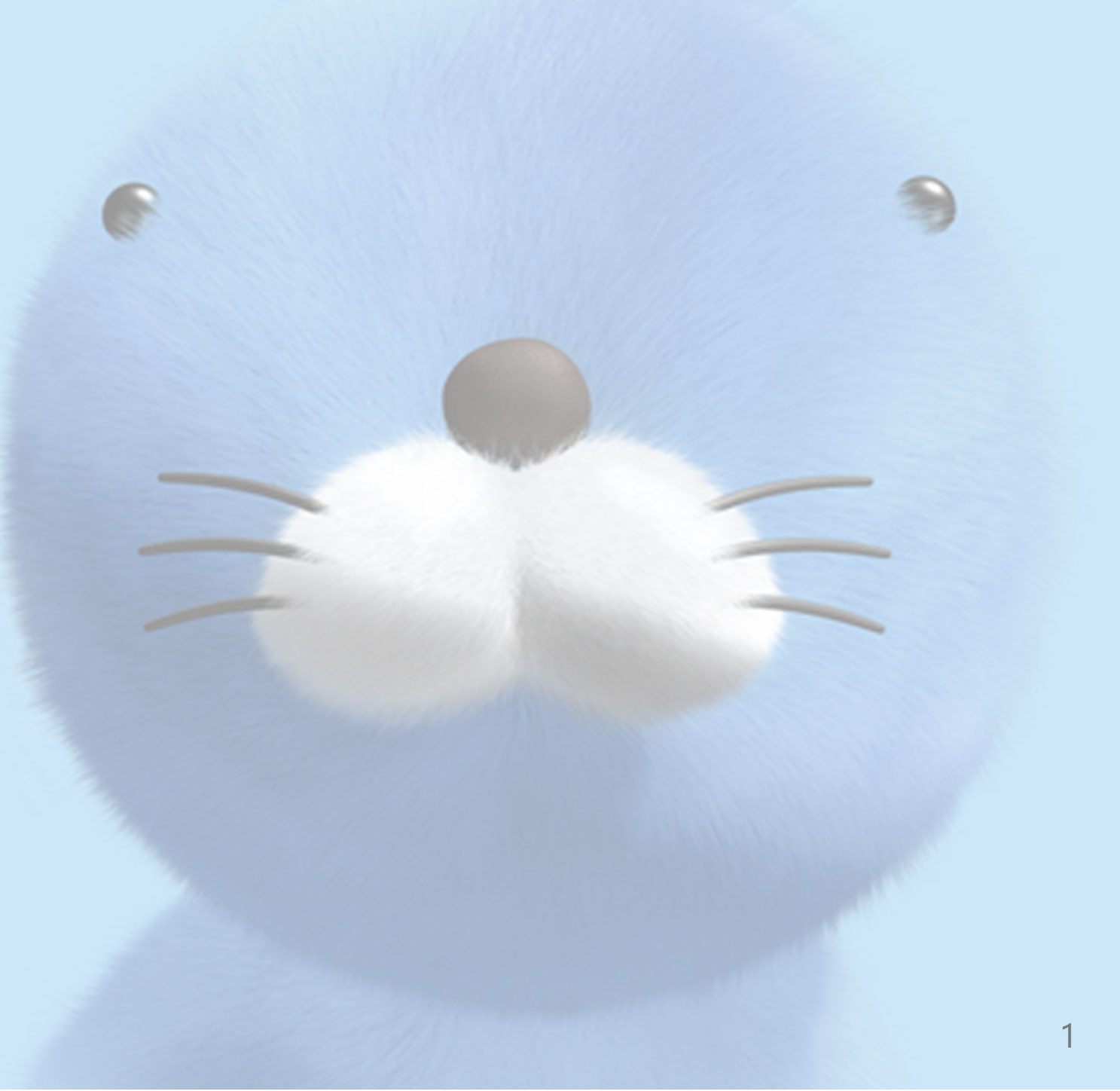
Python 신입 SIG: 자료형

Sep 20th/2017



Python의 자료형

이 중 이 챕터에서는 강조표시된 것만 배울것임.

분류	종류
Numeric Types	`int` , `float` , `complex`
Iterator Types	
Sequence Types	`list` , `tuple` , `range`
Text Sequence Type	`str`
Binary Sequence Types	`bytes` , `bytearray` , `memoryview`
Set Types	`set` , `frozenset`
Mapping Types	`dict`
Other	Boolean, Modules, Classes, Functions, Methods ...

Python 3.6.2 Documentation - Built-in Types

숫자형

- 정수형
  - 정수를 대입하는 자료형
  - 영어 integer를 줄여 `int` 라고 함
  - 정수간 덧셈, 뺄셈, 곱셈의 결과는 정수
  - 정수끼리 나누면 실수(Python 3)
  - 몫을 얻으려면 `//` 사용, 나머지는 `%` 사용
  - 정확한 값을 저장
- 실수형
  - 소숫점 포함된 실수
  - 부동소숫점(floating point) 이용하므로 `float` 라고 함
  - 아주 정확하지는 않음(약간의 오차 있음)

정수형

우리가 사용하는 10진수 외에도 2진수, 8진수, 16진수도 입력 가능

```
a = 123
#      ^ 소숫점 없음.
b = 0b10101100
#   ^^ 0b로 시작. 대소문자 상관 없음.
c = 0o1736 # (=01736)Octal
#   ^^ 0o로 시작. 대소문자 상관 없음.
d = 0xA5f9
#   ^^ 0x로 시작. 대소문자 상관 없음.
```

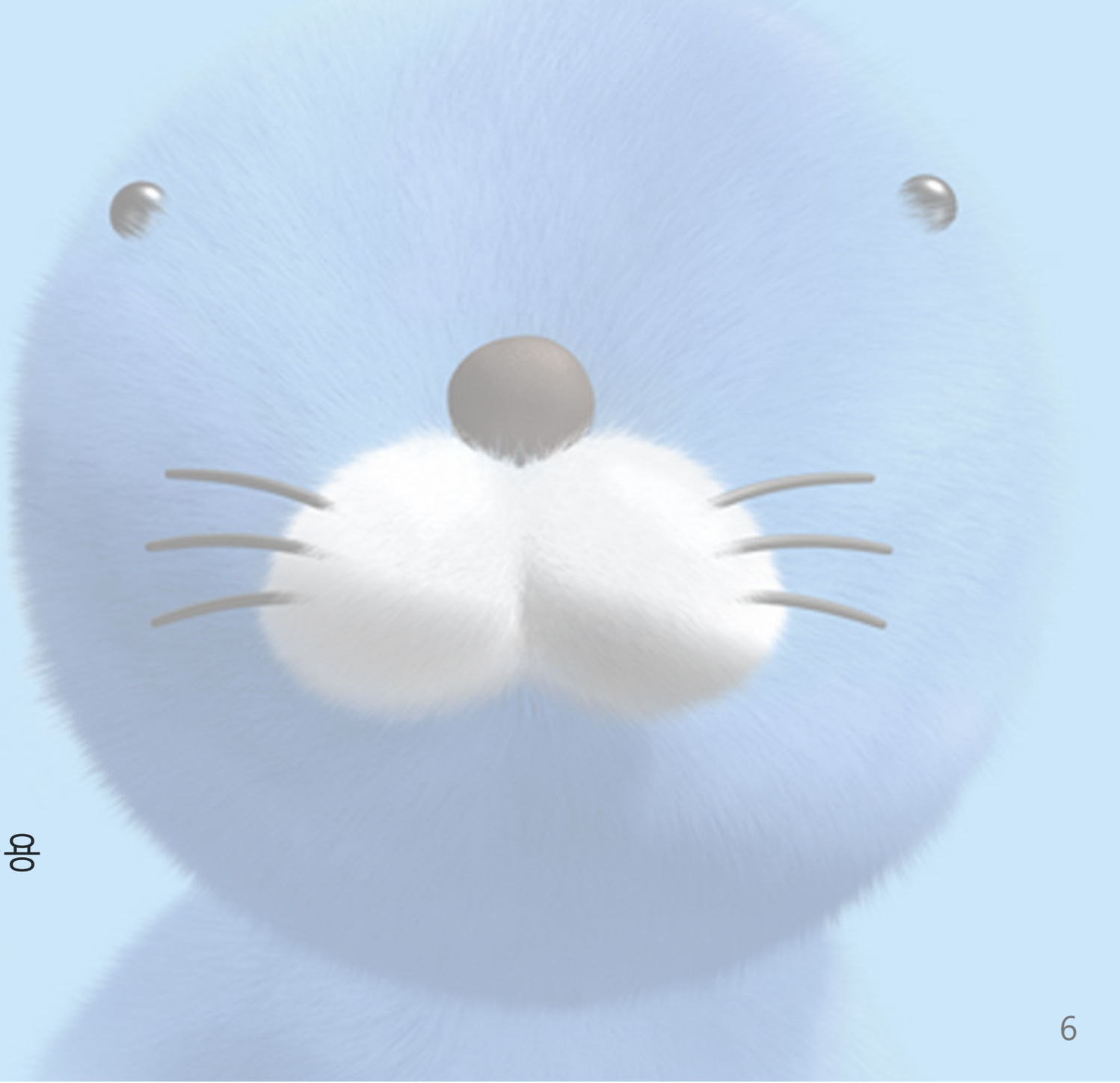
실수형

```
a = 3.1415
#    ^ 소숫점 있음.
b = 6.626E-34
#         ^ 지수 표기법.
```

지수 표기법 `x.xxEyy` 은 $x.xx \times 10^{yy}$을 의미함.

숫자형의 변환

1. int 또는 string을 float로
    - `float()` 사용.
2. float 또는 string을 int로
    - `int()` 사용
3. int나 float를 string으로
    i. int/float -> str
        - `str()` 사용
    ii. int -> 2/8/16진수
        - `bin()`, `oct()`, `hex()` 사용



Example

```
a = 582
b = 26.1284
c = "23"
d = "6.334"

e = float(a)
f = float(d)
g = int(b)
h = int(c)
i = str(a)
j = str(b)
k = hex(a)
l = oct(a)
m = bin(a)
# 다음 슬라이드로
```

Example

```
# 전 슬라이드에서 이어짐
print(e,type(e).__name__)
print(f,type(f).__name__)
print(g,type(g).__name__)
print(h,type(h).__name__)
print(i,type(i).__name__)
print(j,type(j).__name__)
print(k,type(k).__name__)
print(l,type(l).__name__)
print(m,type(m).__name__)
```

출력

```
582.0 float
6.334 float
26 int
23 int
582 str
26.1284 str
0x246 str
0o1106 str
0b1001000110 str
```

String

- 말 그대로 문자열임.
- 만들어지면 내용 변경 불가능하다.
- `+`로 문자열을 이을 수 있고, `*`로 반복할 수 있다.
- `len(<str>)`을 이용하면 길이를 얻을 수 있다.

Single-line

- `'` 또는 `"`로 감싸면 됨.
- 둘 사이 차이는 없으나, `'`로 감싼 문자열 안에는 `"`을, `"`로 감쌀 경우 `'`을 그대로 사용 불가(escape sequence를 이용하면 사용 가능).

Multiline

- `'''` 또는 `"""`로 감싸면 됨
- 줄바꿈 유지
- `'` 또는 `"` 사용 가능

Example

```
str_1 = "'Hello, world!'"
print(str_1)
str_2 = '"Hello, world!"'
print(str_2)
str_3 = '''Line 1
Line 2'''
print(str_3)
str_4 = """Line 3
Line 4"""
print(str_4)
```

Output

```
'Hello, world!'
"Hello, world!"
Line 1
Line 2
Line 3
Line 4
```

Indexing

- 문자열의 특정 위치의 글자를 가르킬때 사용
- index는 0부터 시작.
- 음수 index는 뒤에서부터 위치를 나타냄

```
str1 = "The quick brown fox jumps over the lazy dog."

print(str1[0]) # 첫 번째 글자
print(str1[11]) # 12번째 글자
print(str1[-1]) # 뒤에서 1번째
print(str1[-4]) # 뒤에서 4번째
```

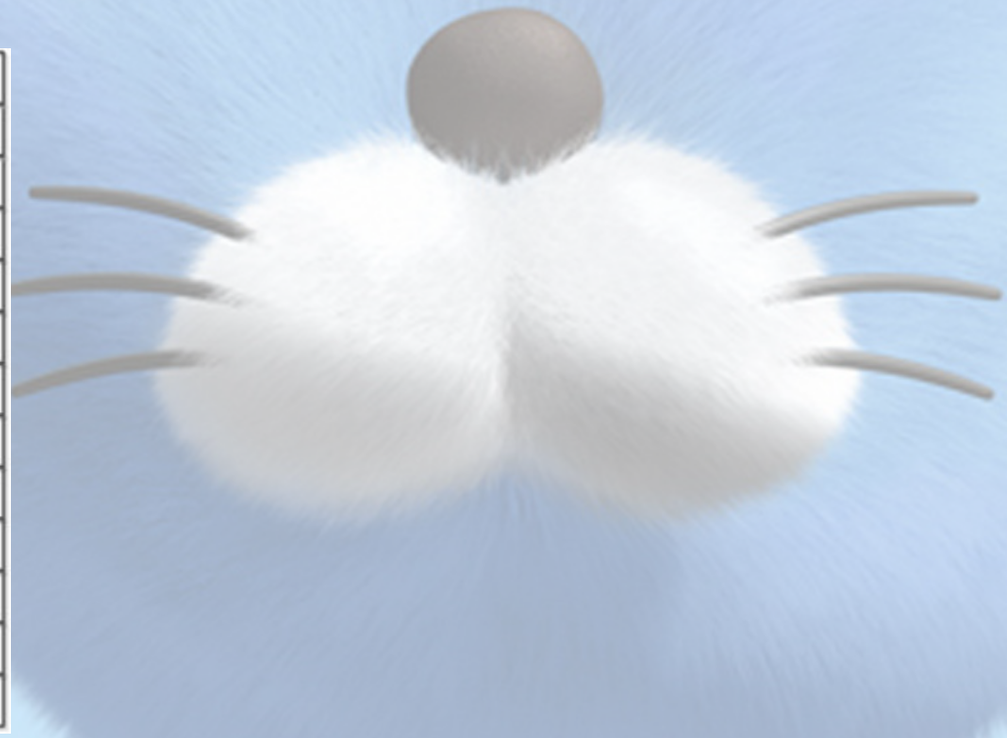
Output

```
T
r
.
d
```

Escape sequence

- 실제로 적힌 문자가 아닌 다른 문자를 나타낸다
- `\` 로 시작

Escape character	Description of Action	Code Used in Programming
\a	Alert sound. A beep is generated by the computer on execution	"\a"
\b	Backspace.	"\b"
\f	Form feed.	"\f"
\n	New line. Shifts the cursor to new line. Words on the right of "\n" go to next line	"\n"
\r	Carriage return. Positions the cursor to the beginning of current line.	"\r"
\t	Horizontal tab, it moves the cursor by a number of spaces or to next tab stop	"\t"
\v	Vertical tab.	"\v"
\\	Backslash. Displays a black slash character (\).	"\ \"
\'	Displays a single-quote character(').	"\'"
\"	Displays a double-quote character(").	"\""
\?	Displays question mark (?).	"\ ?"
\0	Null character. Marks the end of string.	"\0"



- 주로 사용하는 것은 `\n`, `\t`, `\'`, `\"`, `\\` 등임.

문자열 formatting

- 문자열 내에 특정 값들을 예쁘게 넣는다.
- `<string>.format(<var1>, <var2>, ... )` 또는 `<string> % (<var1>, <var2>, ... )` 사용

```
number = 20
welcome = '환영합니다'
base = '{} 번 손님 {}'

#아래 4개의 print는 같은 값을 출력
print(number,'번 손님',welcome)
print(base.format(number,welcome))
print('{} 번 손님 {}'.format(number,welcome))
print('%d 번 손님 %s' % (number,welcome))
```

Output

```
20 번 손님 환영합니다
20 번 손님 환영합니다
20 번 손님 환영합니다
20 번 손님 환영합니다
```

`<string>.format()` 을 이용한 문자열 formatting

- Formatting 할 문자열의 `{}` 로 둘러싸인 자리에 format의 argument를 넣음
- `{}` 안에 숫자로 된 index나 문자로 된 이름을 넣을 수 있음

```
name = '홍길동'
contry = '율도국'
print("{}은 {} 사람이다.".format(name, contry))
print("{0}은 {1} 사람이다.".format(name, contry))
                                  ^ 0    ^ 1
print("{1}은 {0} 사람이다.".format(contry, name)) # index를 통해 순서를 바꿀 수 있다.
                                  ^ 0     ^ 1
print("{nm}은 {ctry} 사람이다.".format(ctry=contry, nm=name))
```

Output

```
홍길동은 율도국 사람이다.
홍길동은 율도국 사람이다.
홍길동은 율도국 사람이다.
홍길동은 율도국 사람이다.
```

`<string> %` 을 이용한 문자열 formatting

- 데이터의 타입에 따라 사용할 format code가 달라짐.

```
date = 12
day = '금'
htemp = 20.32694
ltemp = 28.27156
pprob = 25
print("%d일 %s요일의 날씨는, 최저기온 %.1f도, 최고기온 %.1f도에 강수확률은 %d%%입니다."
 % (date, day, htemp, ltemp, pprob)) # 순서대로 대응
```

Output

```
12일 금요일의 날씨는, 최저기온 20.3도, 최고기온 28.3도에 강수확률은 25%입니다.
```

Format Code

C에서 사용되는 코드와 비슷.

`.format` 사용하는 경우에도 쓸 수 있다(아주 약간 다르다.)

코드	설명
`%s`	String
`%c`	Character
`%d`	Integer
`%f`	Floating point
`%o`	Octal
`%x`	Hexadecimal
`%%`	Character `%`

Python Documentation - Format Specification Mini-Language

Format Code

숫자와 같이 사용하여 정렬, 표시자릿수 조절 등을 할 수 있다.

1. 표시할 영역 크기 조절

```
print(">%6d<" % 12)
#         ^ 6칸을 사용한다.
pritn(">%6d<" % 123456789) # 길이 초과시에는 제한 무시.
```

Output

```
>    12<
>123456789<
```

Format Code

2. 출력값의 정렬

```
print(">%6d<" % 12)
# 오른쪽정렬(기본).
print(">%-6d<" % 12)
#         ^ 왼쪽정렬.
```

Output

```
>    12<
>12    <
```

Format Code

3. 숫자 앞의 0 표시

```
print(">%06d<" % 12)
#         ^ 숫자 앞의 생략된 0을 표시한다.
```

Output

```
>000012<
```

Format Code

4. 소숫점 자릿수(Float)/표시 글자수(String) 조절

```
print(">%6.3f<" % 3.14159265)
#          ^ 소숫점 아래 3자리까지 표시한다.
print(">%.3f<" % 3.14159265)
# 같은 기능. 전체 길이 제한 없이
print(">%8.4s<" % "Hello!")
#          ^ 문자열을 앞에서 4글자까지만 표시
```

Output

```
> 3.142<
>3.142<
>    Hell<
```

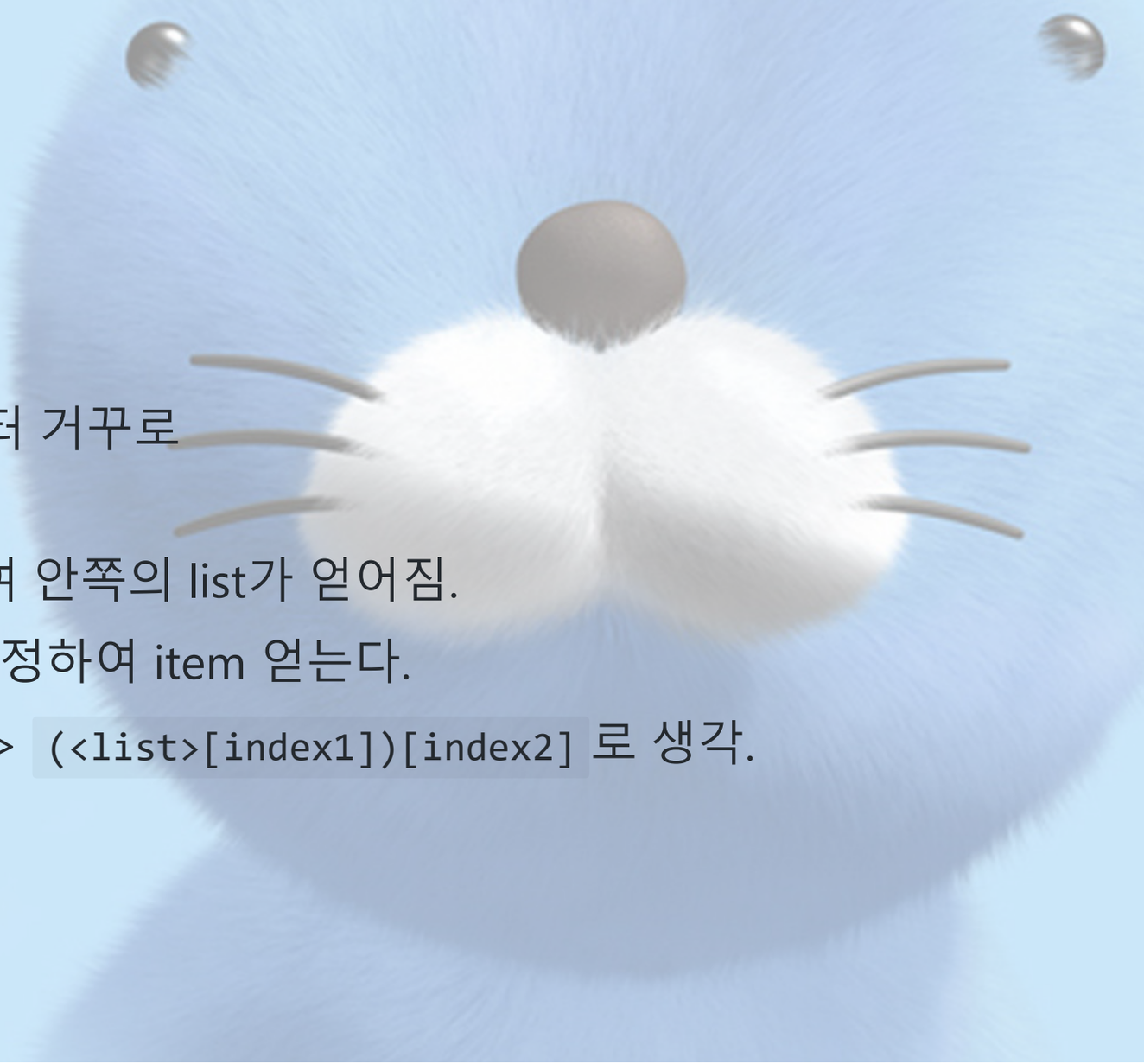
List

- 순서가 있는 값들의 집합
- item들을 `[]`로 감싸고 `,`로 나눈다.
- 빈 list는 `list()`나 `[]`로 생성할 수 있다.
- 안에는 어떤 자료형도 넣을 수 있다.
- 일부 자료형은 list로 변환할 수 있다.

```
a = list() # An empty list
b = [] # Another empty list
c = [1, 2, 'a', 'Hello', 12.2]
d = [1, 2, [3, 4 ,5], 'Hi!']
e = list("Hello!") # == ['H', 'e', 'l', 'l', 'o', '!']
```

Python 3 Documentation - Lists

List의 Indexing



- String과 같다
  - `<list>[index]` e.g. `lst1[3]`
  - 0부터 시작, 음수는 끝에서부터 거꾸로
- List 안의 list
  - 바깥쪽 list에 index를 지정하여 안쪽의 list가 얻어짐.
  - 이 list에 대해 다시 index를 지정하여 item 얻는다.
  - i.e. `<list>[index1][index2]` -> `(<list>[index1])[index2]` 로 생각.

Example

```
lst1 = ['dog', 'cat', 'mouse', 'cow', 'horse']
lst2 = [1, 'a', 'b', [2, 4, 6], 'c']
print(lst1[2]) # 앞에서부터
print(lst1[-1]) # 뒤에서부터
print(lst2[0])
print(lst2[3]) # list안의 list
print(lst2[3][1]) # list안의 list의 item
```

Output

```
mouse
horse
1
[2, 4, 6]
4
```

List Slicing

- 리스트를 나눌 때 사용
- `<list name>[<start>:<end>]` 와 같이 사용한다(end는 포함 x).
- 문자열에서도 사용할 수 있다.
- `start` 나 `end` 는 생략 가능
  - `start` 생략 시 처음부터 slicing, `end` 생략 시 끝까지 slicing 한다.

```
lst1 = ['a','b','c','d','e','f']
print(lst1[1:3])
print(lst1[:2])
print(lst1[3:])
print(lst1[1:-2])
```

Output

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

List 연산과 수정

1. 연산 : 문자열과 같다.
   - `+` : 이어붙이기 (e.g. `[1,2,3]+['a','b','c']` 은 `[1, 2, 3, 'a','b','c']`)
   - `*` : 반복 (e.g. `[1, 2, 3]*2` 은 `[1, 2, 3, 1, 2, 3]`)
2. 수정
   - 한 item 수정: Index로 지정 후 변수처럼 수정. (e.g. `lst1[1]=3.21`)
     - `del` 함수(object 삭제) 이용하여 단일 요소 삭제 가능. (e.g. `del lst1[3]`)
   - 연속된 범위 수정: Slicing 이용. (e.g. `lst2[1:3]=['r', 's', 't']`)
     - 빈 리스트 이용해 해당 범위 삭제 가능. (e.g. `lst2[3:5]=[]`)

       -특정 값 삭제: `<list>.remove(<value>)` 이용하여 해당 값 삭제(여러개면 가장 앞의 item)

Example

```
lst1 = ['a','b','c','d','e','a']
lst2 = [1,2,3]
print(lst1+lst2)
print(lst2*2)
lst1[2] = 3.14159
print(lst1)
lst1.remove('a')
print(lst1)
del lst1[1]
print(lst1)
lst1[2:4] = ['s','t','u']
print(lst1)
lst1[1:3] = []
print(lst1)
```

Example

Output

```
['a', 'b', 'c', 'd', 'e', 'a', 1, 2, 3]
[1, 2, 3, 1, 2, 3]
['a', 'b', 3.14159, 'd', 'e', 'a']
['b', 3.14159, 'd', 'e', 'a']
['b', 'd', 'e', 'a']
['b', 'd', 's', 't', 'u']
['b', 't', 'u']
```

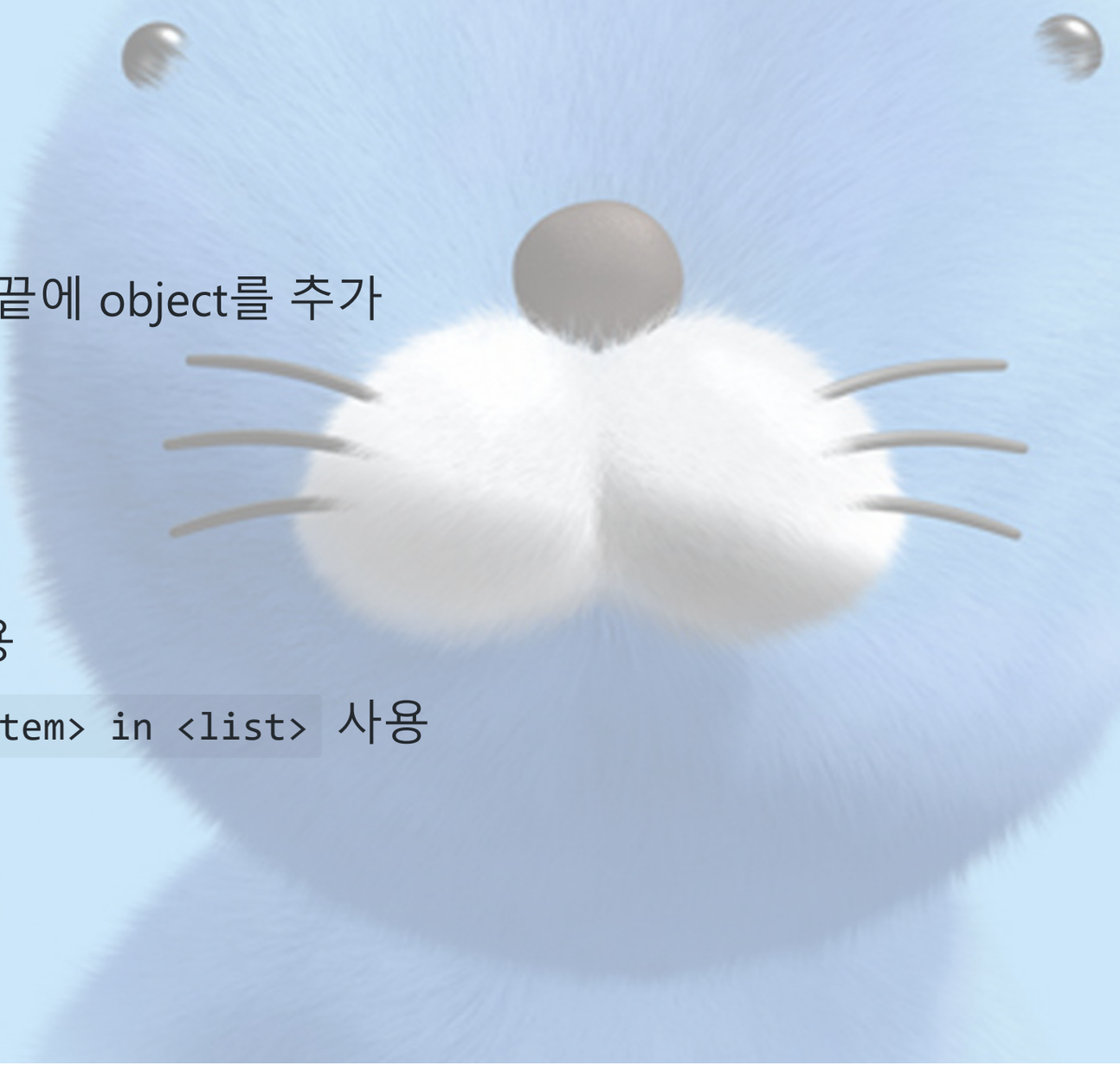
List의 연산과 수정

3. item 추가

- `<list.>append(object)` 사용 : List 끝에 object를 추가

4. 정렬 : `<list.>sort()` 사용
5. 뒤집기 : `<list.>reverse()` 사용
6. 길이 확인 : `len()` 사용
7. 전부 지우기 : `<list>.clear()` 사용
8. 해당 값이 원소로 있는지 확인 : `<item> in <list>` 사용

Python 3 Documentation - More on Lists



Example

```
lst1 = ['d','c','a','b','f','e']
lst1.append('z')
print(lst1)
lst1.reverse()
print(lst1)
lst1.sort()
print(lst1)
print(len(lst1))
lst1.clear()
print(lst1)
```

Output

```
['d', 'c', 'a', 'b', 'f', 'e', 'z']
['z', 'e', 'f', 'b', 'a', 'c', 'd']
['a', 'b', 'c', 'd', 'e', 'f', 'z']
7
[]
```

List Comprehension

- List에서 list를 만든다
- 수학의 집합에서 $S' = \{ f(x) \mid x \in S,\ x^2 > 3 \}$와 비슷하다.
- `[ <output expression> for <variable> in <iteraterble obj> if <predicate>]` 또는 `[ <output expression> for <variable> in <iteraterble obj>]`
  (e.g. `[2*x for x in lst_1 if x>1.5]` : lst_1 안의 원소 x에 대해 x가 1.5 이상이면 x에 2배를 하여 list를 만든다.)

Python 3 Documentation - List Comprehensions

Example

```
lst1 = [72, 101, 108, 108, 111, 44, 32, 87, 111, 114, 108, 100, 33]
lst2 = [chr(x) for x in lst1] # List 안의 정수를 해당되는 글자로 변환
# (chr(x) : x에 해당하는 코드값을 가진 글자 반환)
print(lst2)
lst3 = [x for x in lst2 if x > 'd'] # 'd'보다 코드값이 큰 글자만
print(lst3)
```

Output

```
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
['e', 'l', 'l', 'o', 'o', 'r', 'l']
```

Tuple

- List와 유사하나 값의 변경이 불가(Immutable)
- `()`로 둘러싸여 있고 값들을 `,`로 나눔
- `()` 생략 가능, entry 1개짜리 만들 때에는 item 뒤 `,` 필요
- 나머지는 list와 거의 동일.
  - Indexing, slicing, `+`, `*` 연산 가능
- `tuple()` 통해 list를 tuple로 바꿀 수 있다.

Python 3 Documentation - Tuple

Example

```
a = (23,'Dog',7.12,[1,2,3],"Hi",(1,2,3))
b = 'a','b',1,2 # () 생략가능
c = 43.12, # 1개짜리
lst = [54,23,8]
print(a[2]) # indexing
print(a+b) # +연산
print(a[1:3]) # slicing
print(b*2) # *연산
print(tuple(lst)) # list -> tuple
```

Output

```
7.12
(23, 'Dog', 7.12, [1, 2, 3], 'Hi', (1, 2, 3), 'a', 'b', 1, 2)
('Dog', 7.12)
('a', 'b', 1, 2, 'a', 'b', 1, 2)
(54, 23, 8)
```

Dictionary

- Key와 value의 조합으로 값들을 저장
- 순차적으로 값을 얻지 않고 key를 통해 얻는다.
- `{}`으로 `:`로 나뉜 key와 value의 순서쌍(`<key>:<value>`)들을 감싼다
- 빈 dictionary는 `dict()`나 `{}`로 생성할 수 있다.
- Key로는 string, numeric types, string이나 numeric type으로만 이루어진 tuple이 가능
  - 정확히는 immutable(생성 후 변경 불가)한 object만 가능
- Key는 unique해야함.
- Value에는 어떤 자료형도 넣을 수 있다.

```
a = dict() # An empty dictionary
b = {} # Another empty dictionary
c = {"AAPL":159.88, "SBUX":54.62, "NKE":53.33, "YHOO":54.02}
#     ^ Key   ^ Value
d = {123:"1", 4.24:"2", (3.4,2.1):"3"} # key로 정수, 실수, tuple 가능
#     ^ Key     ^ Key       ^ Key
```

Python 3 Documentation - Dictionaries

Dictionary

- Dictionary의 값은 key로 접근
- List의 index 대신 key를 넣는다 (e.g. `<dictionary>[<key>]`)
- `get`을 사용하면 키가 없을 경우 None을 반환
- `<key> in <dictionary>`를 이용하여 해당 key가 있는지 확인 가능

```
stocks = {"AAPL":159.88, "SBUX":54.62, "NKE":53.33, "YHOO":54.02}
print(stocks["AAPL"])
# print(stocks[0]) # KeyError. 0이라는 key가 없음.
print(stocks.get(0)) # None 반환
```

Output

```
159.88
None
```

Dictionary의 수정

1. 추가
   `<dictionary>[<key>]=<value>` 와 같이 한다.
2. 수정
   추가와 같다.
3. 삭제
   - List에서 사용한 `del` 을 사용
     `del(<dictionary>[<key>])`
   - 전부 지우려면 `<dictionary>.clear()` 사용

Example

```
stocks = {"AAPL":159.88, "SBUX":54.62, "NKE":53.33, "YHOO":54.02}
stocks["GOOG"]=921.81
print(stocks)
stocks["AAPL"]=158.73
print(stocks)
del(stocks["NKE"])
print(stocks)
```

Output

```
{'SBUX': 54.62, 'GOOG': 921.81, 'YHOO': 54.02, 'NKE': 53.33, 'AAPL': 159.88}
{'SBUX': 54.62, 'GOOG': 921.81, 'YHOO': 54.02, 'NKE': 53.33, 'AAPL': 158.73}
{'SBUX': 54.62, 'GOOG': 921.81, 'YHOO': 54.02, 'AAPL': 158.73}
```

Dictionary에서 key/value 얻기

1. Key들의 list : `<dictionary>.keys()`
2. Value들의 list : `<dictionary>.values()`
3. Key와 value의 쌍(tuple) : `<dictionary>.items()`

   정확히는 list가 아니라 view임.

```
stocks = {"AAPL":159.88, "SBUX":54.62, "NKE":53.33, "YHOO":54.02}
print(list(stocks.keys()))
print(list(stocks.values()))
print(list(stocks.items()))
```

Output

```
['SBUX', 'YHOO', 'NKE', 'AAPL']
[54.62, 54.02, 53.33, 159.88]
[('SBUX', 54.62), ('YHOO', 54.02), ('NKE', 53.33), ('AAPL', 159.88)]
```