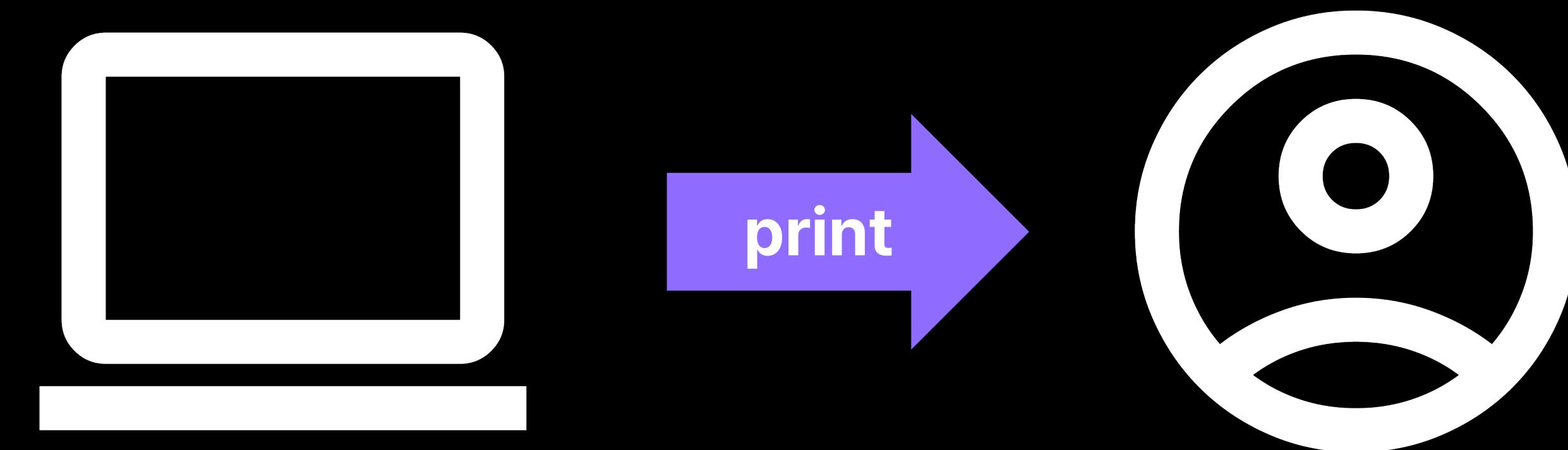


Python 기초 연산과 자료형



Python에서는 우리가 원하는 정보나 자료를
컴퓨터가 **출력**하게 할 수 있습니다!



컴퓨터

사용자



출력

```
print("출력할 내용")
```

```
print("나의 꿈은 파이썬 정복!")
```

실행 결과

나의 꿈은 파이썬 정복!



Q. 하나가 아닌, 여러 자료를 출력하고 싶은데,
어떻게 하면 좋을까?

실행 결과

3 Hello!



A. ,(콤마)를 이용해서 여러 자료를 출력할 수 있다!

```
print(3, "Hello")
```

실행 결과

3 Hello!



Q. print로 삼행시를 짓고 싶은데...
어떻게 하면 좋을까?

실행 결과

비행기에 타신 승객 여러분
행복한 여행 되십시오
기내식은 바밤바



A. print를 여러 번 사용하자!

```
print("비행기에 타신 승객 여러분")
```

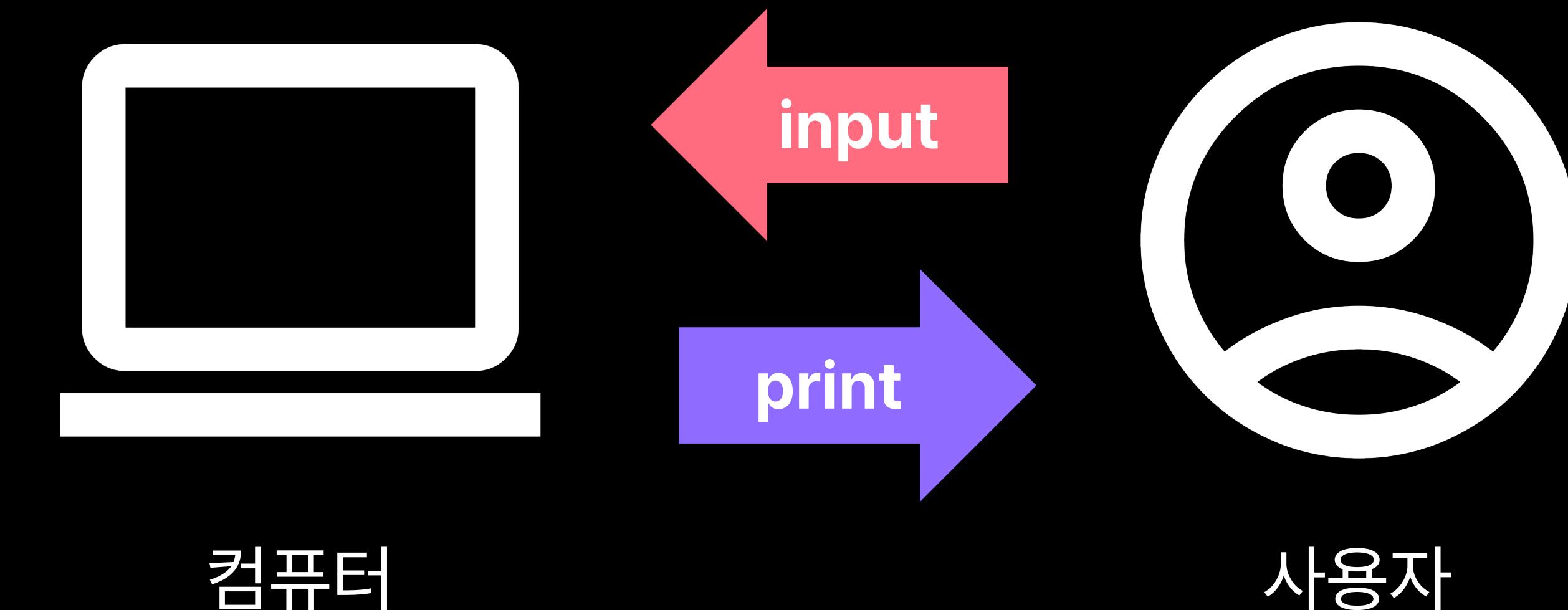
```
print("행복한 여행 되십시오")
```

```
print("기내식은 바밤바")
```



입력

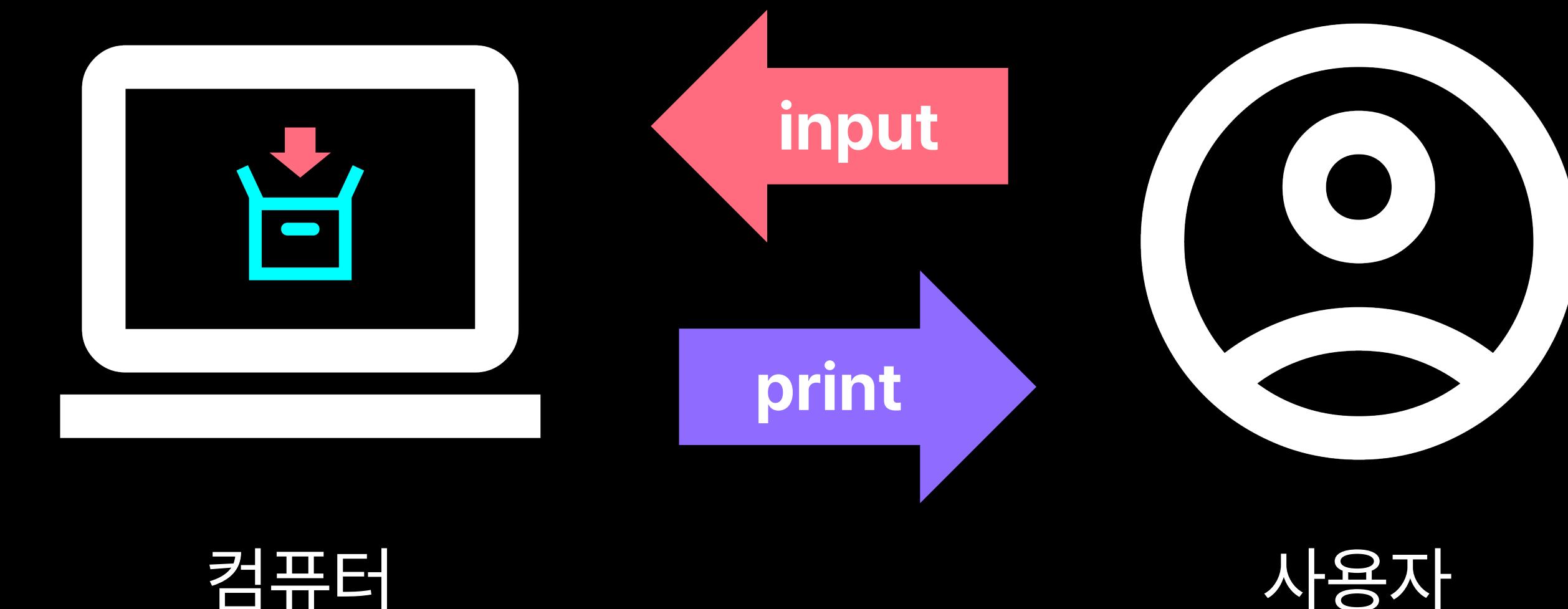
print를 이용해서 컴퓨터로부터 정보를 받았다면,
정보를 컴퓨터에게 전달할 수 있음





입력

컴퓨터는 사용자가 전달한 값(자료)을 **어딘가에 보관**해야 함
→ **변수**를 사용!





input()

변수 = `input()`

터미널에 값을 “**입력**” + Enter

```
var = input()
```

실행 결과

/* 코드가 실행되는 중입니다! */

> 터미널 입력 값을 넣어주세요.



input()의 중요한 특징

무엇을 입력하든 “문자열”로 입력 받아짐

```
var = input("숫자를 입력하세요 : ")  
  
print(var)  
print(type(var))
```

실행 결과

```
숫자를 입력하세요 : 10  
10  
<class 'str'>
```



input()의 중요한 특징

입력 받은 자료를 자유롭게 사용하기 위해서는 **가공**이 필요

```
var1 = input("숫자를 입력하세요 : ")
var2 = input("숫자를 입력하세요 : ")

print(var1 + var2)
```

실행 결과

```
숫자를 입력하세요 : 100
숫자를 입력하세요 : 300
100300
```



어떻게 가공해 주지?

만약, 문자열을 숫자로 바꾸고 싶다면? → **자료형을 변환(형 변환)**을 사용

```
var1 = "8"  
var2 = "1000"  
print(var1 + var2)  
print(int(var1) + int(var2))
```

실행 결과

```
81000  
1008
```



형 변환

구분	이름	예시	결과
정수	<code>integer</code>	<code>int("1008")</code>	1008
실수	<code>float</code>	<code>float("3.14159")</code>	3.14159
문자열	<code>string</code>	<code>str(3.14159)</code>	"3.14159"
리스트	<code>list</code>	<code>list("1008")</code>	["1", "0", "0", "8"]



형변환

```
a = "345"  
b = int("345")
```

```
print(a, b) # 345 345
```

```
print(type(a)) # <class 'str'> | 문자열
```

```
print(type(b)) # <class 'int'> | 숫자
```



숫자형 (Number)

3 # 정수(integer)

3.14 # 실수(float)

3+4j # 복소수

- 숫자로 이루어진 자료형
- 정수나 실수 등을 다룰 수 있음
- 숫자끼리의 연산 가능



문자열 (String)

'Hello!'

'3.14' # 작은 따옴표 OK

"3.14" # 큰 따옴표 OK

- 문자나 문자들을 늘어놓은 것
- **큰 따옴표(" ")** 와 **작은 따옴표('')**로 구분



str.split(c)

c를 기준으로 문자열을 쪼개서 리스트 형태로 반환
(괄호를 비울 시 공백을 기준으로 처리)

```
my_str = "1 2 3 4 5"  
print(my_str.split())  
  
element = "Na,Mg,Al,Si"  
print(element.split(","))
```

실행 결과

```
["1", "2", "3", "4", "5"]  
["Na", "Mg", "Al", "Si"]
```



str.join(list)

str을 기준으로 리스트를 합쳐서 문자열을 반환

```
my_list = ["e", "l", "i", "c", "e"]
print("".join(my_list))

station = ["서울", "대전", "대구"]
print("→".join(station))
```

실행 결과

elice

서울→대전→대구



리스트 (List)

```
[] # 빈 리스트
```

```
['a', 'b']
```

```
# 다른 자료형을 함께!
```

```
['a', 2]
```

- 여러 자료를 보관하는 자료형
- 다른 종류의 자료를 함께 담을 수 있음
- 자료 안에 순서 존재



주석 (Comment)

주석 처리한 말들은

"""

컴퓨터가 실행하지

않아요!

"""

- 주석은 컴퓨터가 무시
- 한 줄 주석은 #
- 여러 줄 주석은 """ """ 또는 ''' ''' 를 사용
(따옴표 세 번)



여러 자료를 담는 자료형이 필요하다면?

대부분 여러 자료를 담기 위해서 리스트 자료형을 사용
하지만 리스트 자료형은 원소가 바뀔 위험이 있음

```
my_list = ["e", "l", "i", "c", "e"]
print(my_list)
```

```
my_list[3] = "ERROR"
print(my_list)
```

실행 결과

```
["e", "l", "i", "c", "e"]
["e", "l", "i", "ERROR", "e"]
```



튜플의 필요성

값을 바꿀 수 없으면서 여러 자료를 담고 싶을 때는
튜플(Tuple) 자료형을 사용!



튜플 (Tuple)

여러 자료를 함께 담을 수 있는 자료형
소괄호()를 이용하여 표현

```
tuple_zero = ()  
tuple_one = (1,)  
tuple_sample = (1, 2, 3, 4, 5)  
  
print(tuple_sample)  
print(type(tuple_sample))
```

실행 결과

```
(1, 2, 3, 4, 5)  
<class 'tuple'>
```



튜플의 특징 (1)

튜플도 Index를 이용한 인덱싱, 슬라이싱 가능

```
my_tuple = ("A", "C", "E")  
  
print(my_tuple[1])  
  
print(my_tuple[:2])
```

실행 결과

C
("A", "C")



튜플의 특징 (2)

in 연산자로 Tuple 안에 원소 확인 가능

len 연산자로 Tuple의 길이 확인 가능

```
my_tuple = ("A", "C", "E")  
  
print("A" in my_tuple)  
  
print(len(my_tuple))
```

실행 결과

True
3



튜플의 특징 (3)

+ 연산자로 Tuple과 Tuple을 연결할 수 있음

* 연산자와 숫자를 이용하면 Tuple을 반복할 수 있음

```
tuple_1 = ("G", "O")
tuple_2 = ("O", "D")
print(tuple_1 + tuple_2)
```

```
print(tuple_1 * 3)
```

실행 결과

```
("G", "O", "O", "D")
("G", "O", "G", "O", "G", "O")
```



튜플의 특징 (4)

튜플은 새로운 자료를 추가하거나 기존의 자료 삭제, 변경이 불가능
한 번 만들어지면 고정되는 자료

```
my_tuple = ("e", "l", "i", "c", "e")
print(my_tuple.append("!"))
print(my_tuple.remove("e"))
my_tuple[1] = "ERROR"
```

실행 결과

```
Traceback (most recent call last):
File "main.py", line 3, in <module>
    print(my_tuple.append("!"))
AttributeError: 'tuple' object has no
attribute 'append'
```



Dictionary?

Dictionary → 사전

사전에서 “단어”와 “뜻”이 하나의 짹꿍인 것처럼 짹꿍이 있는 자료형

dictionary

noun • UK /'dɪkʃnəri/ US /'dɪkʃəneri/ PLURAL dictionaries

★ A1 a book that contains a list of words in alphabetical order with their meanings explained and sometimes written in another language
[사전](#)
Use your dictionaries to look up any words you don't understand.

성	이름
이메일	
비밀번호	
비밀번호 확인	



Dictionary (딕셔너리)

중괄호와 콜론(:)을 이용하여 표현

```
dictionary_zero = {}  
person = {"name" : "Rabbit", "age" : 22}  
  
print(person)
```

실행 결과

```
{"name" : "Rabbit", "age" : 22}
```



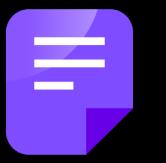
Dictionary (딕셔너리)

짝꿍이 있는 자료형이므로
Key를 알면 Value를 알 수 있음

```
dictionary_zero = {}  
person = {"name": "Rabbit", "age": 22}  
  
print(person)
```

실행 결과

```
{"name": "Rabbit", "age": 22}
```

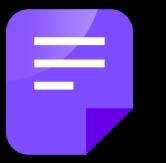


열쇠처럼 딕셔너리에서 자료를 꺼낼 수 있는 도구

```
dictionary_zero = {}  
person = {"name": "Rabbit", "age": 22}  
  
print(person)
```

실행 결과

```
{"name": "Rabbit", "age": 22}
```



딕셔너리에서 Key를 이용하여 꺼낸 자료

```
dictionary_zero = {}  
person = {"name": "Rabbit", "age": 22}  
  
print(person)
```

실행 결과

```
{"name": "Rabbit", "age": 22}
```



딕셔너리에서 Key를 이용하여 자료(value) 꺼내기

```
person = {"name": "Rabbit", "age": 22}  
  
print(person["name"])  
print(person["age"])
```

실행 결과

Rabbit
22



딕셔너리에서 자료 추가하기

```
person = {  
    "name": "Rabbit",  
    "age": 22  
}  
  
person["mail"] = "rabbit@elice.com"  
  
print(person)
```

실행 결과

```
{"name": "Rabbit",  
 "age": 22,  
 "mail": "rabbit@elice.com"}
```



del을 이용하면 Dictionary의 원소(Key와 Value) 삭제 가능

```
person = {  
    "name": "Rabbit",  
    "age": 22,  
    "mail": "rabbit@elice.com"  
}
```

```
del person["age"]
```

```
print(person)
```

실행 결과

```
{"name": "Rabbit",  
 "mail": "rabbit@elice.com"}
```



keys() 를 이용하면 Dictionary 의 key 들만 추출 가능

```
person = {  
    "name" : "Rabbit",  
    "age" : 22,  
    "mail" : "rabbit@elice.com"  
}  
  
print(person.keys())
```

실행 결과

```
dict_keys(['name', 'age', 'mail'])
```



values() 를 이용하면 Dictionary 의 value 들만 추출 가능

```
person = {  
    "name" : "Rabbit",  
    "age" : 22,  
    "mail" : "rabbit@elice.com"  
}  
  
print(person.values())
```

실행 결과

```
dict_values(['Rabbit', 22,  
'rabbit@elice.com'])
```



items()를 이용하면 Dictionary의 Key와 Value의 쌍 추출 가능

```
person = {  
    "name" : "Rabbit",  
    "age" : 22,  
    "mail" : "rabbit@elice.com"  
}  
  
print(person.items())
```

실행 결과

```
dict_items([('name', 'Rabbit'), ('age', 22),  
           ('mail', 'rabbit@elice.com')])
```



Key는 변할 수 없는 자료형이어야 함
→ 리스트는 사용할 수 없고 튜플은 사용할 수 있음

```
dictionary_example = {[1, 2, 3] : "Hello"}  
  
print(dictionary_example)
```

실행 결과

```
dict_items([('name', 'Traceback (most recent call last):  
  File "main.py", line 1, in <module>  
    dictionary_example = {[1, 2, 3] : "Hello"}'])
```

```
TypeError: unhashable type: 'list'  
Rabbit'), ('age', 22),  
('mail', 'rabbit@elice.com')])
```

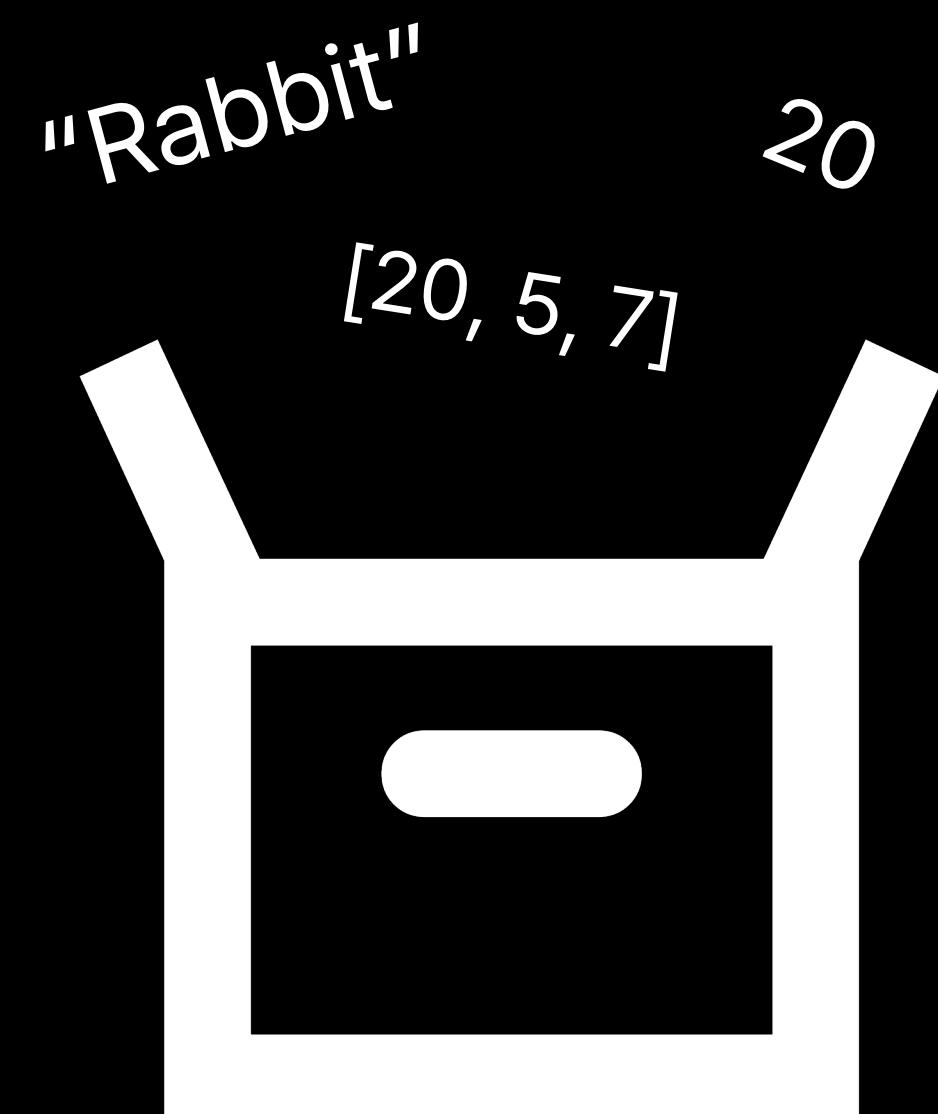
```
dictionary_example = {(1, 2, 3) : "Hello"}  
  
print(dictionary_example)
```

실행 결과

```
{(1, 2, 3) : "Hello"}
```



자료를 “그릇”에 담아서 보관하고 사용하면 편리,
변수(Variable)이라고 함





변수 이름 = 자료

```
num = 10 # 숫자  
name = "Michael" # 문자열  
grade = ['A+', 'B+', 'A0'] # 리스트
```



변수 이름 짓는 법

테스트 자동화 기초

Python 기초 및 데이터
처리

Python 기초 연산과 자
료형

변수 이름에는 숫자, 알파벳, 한글, 언더바(_) 등을 사용할 수 있음

Case 1 : 알파벳 (권장)

```
python = "파이썬"
```

```
print(python)
```

Case 2 : 한글

```
파이썬 = "Python"
```

```
print(파이썬)
```

Case 3 : 알파벳 + 언더바(_) + 숫자

```
carrot_123 = 1234
```

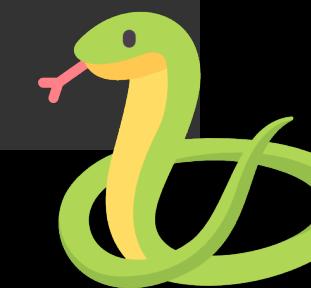
```
print(carrot_123)
```



변수 이름을 짓는 대표적인 방법으로
스네이크 표기법과 **카멜 표기법**이 있음

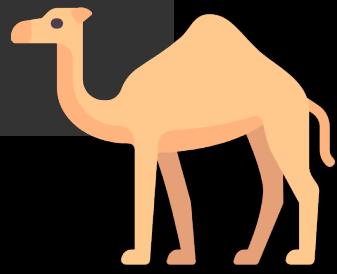
스네이크 표기법

```
prod_name = "milk"  
prod_date = "20**-10-08"
```



카멜 표기법

```
prodName = "milk"  
prodDate = "20**-10-08"
```





```
1name = "Coke"
```

```
print(1name)
```

```
1234 = 5678
```

```
print(1234)
```

```
print = "Hello"
```

```
print(print)
```

```
prod name = "milk"
```

```
prod+date = "20**-10-08"
```

```
print(prod name, prod+date)
```

1. 변수 이름이 숫자로 시작하면 안 됨
2. 숫자로만 구성된 변수 이름 금지
3. 파이썬 문법에서 사용하는 예약어 사용 금지
(e.g. print 등)
4. 공백 문자와 연산자 사용 금지
(e.g. +, -, % 등)

 $+$ $-$ \times \div

더하기

빼기

곱하기

나누기



+

-

*

/

더하기

빼기

곱하기

나누기



```
print(3 + 5) # 8
```

```
print(3 - 5) # -2
```

```
print(3 * 5) # 15
```

```
print(3 / 5) # 0.6
```



///

-

* *

몫 연산자

나머지 연산자

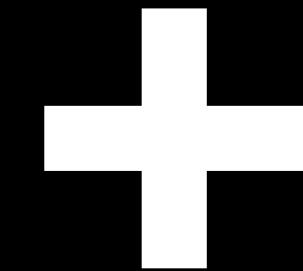
제곱 연산자



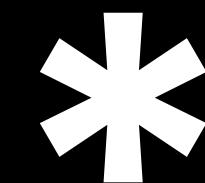
```
print(13 // 5) # 2
```

```
print(13 % 5) # 3
```

```
print(2 ** 4) # 16
```



이어붙이기
(with 문자열)



반복하기
(with 숫자)



문자형 자료의 연산

테스트 자동화 기초

Python 기초 및 데이터
처리

Python 기초 연산과 자
료형

```
print("안녕" + "하세요")  
# 안녕하세요
```

```
print("안녕" * 3)  
# 안녕안녕안녕
```



[질문 1]

“rescue” 와 “secure”은 다른 문자열이다.
이유는 무엇일까요?

[질문 2]

[1, 2, 3] 과 [3, 2, 1] 은 다른 리스트이다
이유는 무엇일까요?

원소의 배치 순서가 다르기 때문



인덱스 (index)

테스트 자동화 기초

Python 기초 및 데이터
처리

Python 기초 연산과 자
료형

문자열과 리스트 자료형은 여러 원소로 이루어져 있고
각각의 위치를 0부터 순서대로 매길 수 있음

“Ready”
01234

[2, 4, 6, 8]
0 1 2 3

“Hi! elice!” #문자열에서는 공백문자도 인덱스에 포함
0123456789



index를 이용해서 문자열이나 리스트의 특정 위치의 원소를 가져오는 방법

String/List[index_number]

```
# alpha에서 인덱스가 1인 원소, 'e'를 출력  
alpha = "Ready"
```

```
print(alpha[1])
```



index를 이용해서 리스트나 문자열의 **일부분**을 가져오는 방법

String/List [**a**_(시작 인덱스) : **b**_(종료 인덱스)]

```
# beta에서 2번째 원소 이상, 5번째 원소 미만을 가져온다
```

```
beta = [2, 4, 6, 8, 10, 12, 14]
```

```
print(beta[2:5])
```

```
# [6, 8, 10]
```