



## 04. 데이터 다루기 - 수와 텍스트, 그리고 비트

뇌를 자극하는 파이썬3

# Contents

## ❖ 변수

## ❖ 수 다루기

- 정수
- 실수
- 복소수
- Math 모듈을 이용한 계산

## ❖ 텍스트 다루기

- 문자열 메소드

## ❖ 수와 텍스트 변환

## ❖ 비트 다루기

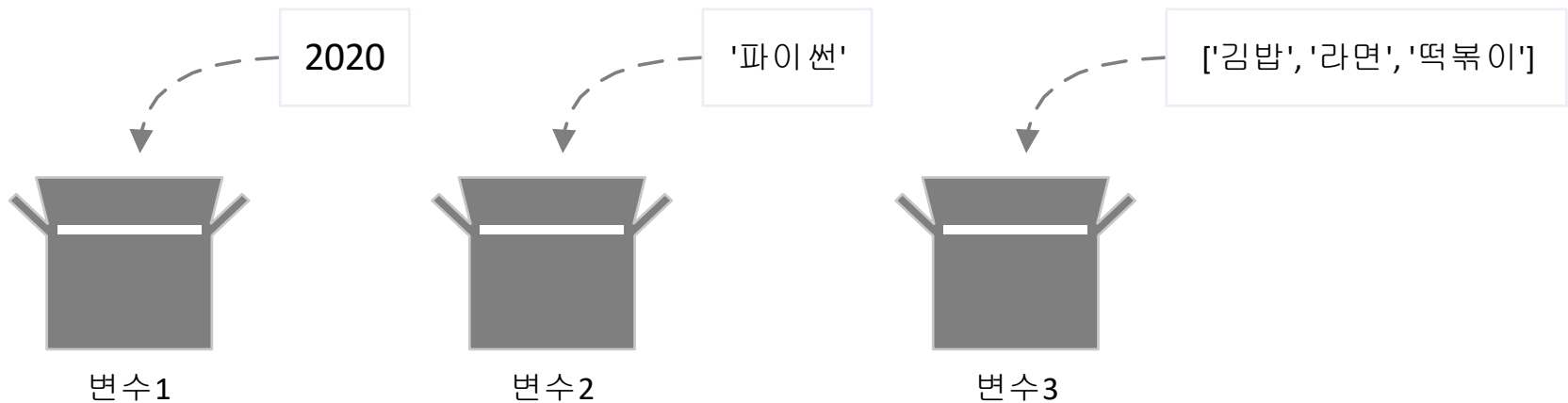
- 시프트 연산자
- 비트 논리 연산자



# 변수

## ❖ 변수란?

- 변수는 데이터를 담는 메모리 공간
- 변수에는 수, 텍스트, 목록, 이미지 데이터 등을 담을 수 있음.



# 변수

## ❖ 실습 1

Type "copyright", "credits" or "license()" for  
formation.

```
>>> a = 2020
```

```
>>> a
```

```
2020
```

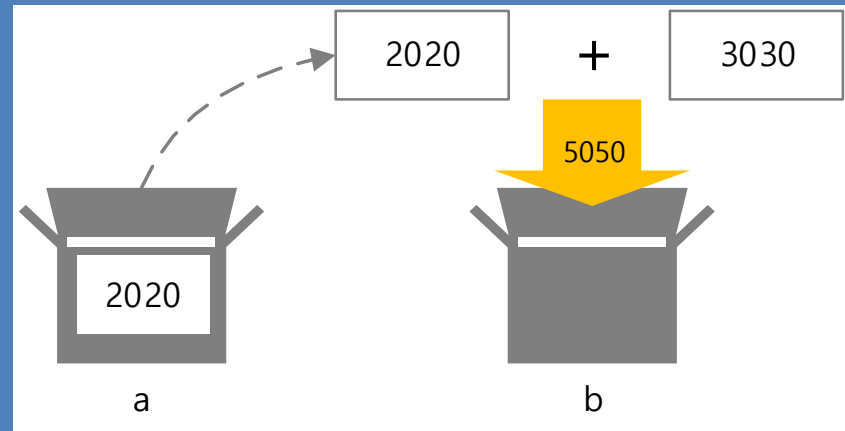
```
>>> b = a + 3030
```

```
>>> b
```

```
5050
```

```
>>> |
```

b = a + 3030 코드에서 파이썬은 a에 담겨있는 2020을 3030과 더한 후 변수 b에 저장



# 변수



## ❖ 실습 2

Type "copyright", "credits" or "license()" for more  
formation.

```
>>> a = 2020
```

```
>>> a
```

```
2020
```

```
>>> b = a + 3030
```

```
>>> b
```

```
5050
```

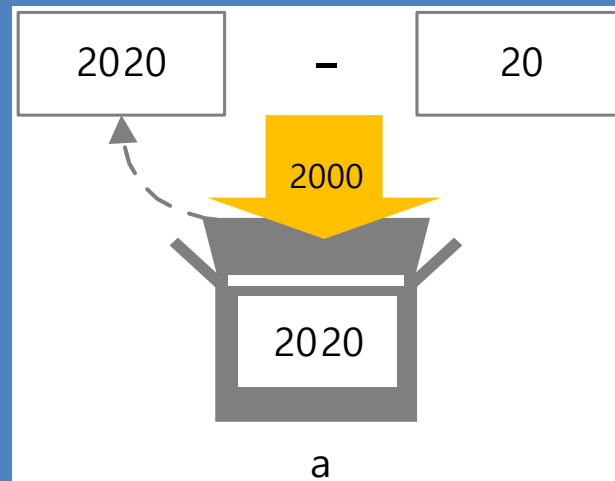
```
>>> a = a - 20
```

```
>>> a
```

```
2000
```

```
>>> |
```

a에 담겨있는 2020에서 20을 뺀 값을 다시 a에 저장



Ln



# 수 다루기

- ❖ 텍스트, 이미지, 음성 등 컴퓨터의 다양한 데이터 처리는 수 처리를 응용하여 이루어짐
- ❖ 파이썬은 기본적으로 지원하는 세 종류의 수
  - 정수
  - 실수
  - 복소수



# 수 다루기 - 정수

## ❖ 정수?

- 음의 정수, 0, 양의 정수
- 파이썬에서는 메모리가 허용하는 한, 무한대의 정수를 다룰 수 있음.

## ❖ 실습 1 (변수에 정수 입력)

```
>>> a = 3
>>> b = 123456789
>>> c = 1234567890123456789012345678901234567890
>>> a
3
>>> b
123456789
>>> c
1234567890123456789012345678901234567890
```



## ❖ 실습 2 (변수에 음수 입력)

```
>>> f = -3  
>>> type(f)  
<class 'int'>
```

```
>>> d = -1234567890123456789012345678901234567890 #  
1234567890을 네 번 타입  
>>> e = -123456789  
>>> f = -3  
>>> d  
-1234567890123456789012345678901234567890  
>>> e  
-123456789  
>>> f  
-3
```





# 수 다루기 - 정수

## ❖ 파이썬에서 제공하는 사칙 연산자

연산	기호
더하기	+
빼기	-
곱하기	*
나눗셈의 몫 구하기	//
나눗셈의 나머지 구하기	%
나누기	/

## ❖ 실습 4 (덧셈과 뺄셈)

```
>>> a = 3 + 4
>>> a
7
>>> b = 7 - 10
>>> b
-3
```



# 수 다루기 - 정수

## ❖ 실습 5 (곱셈)

```
>>> c = 7 * -3  
>>> c  
-21
```

## ❖ 실습 6 (나눗셈의 몫 연산과 나머지 연산)

```
>>> d = 30 // 7  
>>> d  
4  
>>> e = 30 % 7 # 30  
>>> e  
2
```

```
>>> d = 30 / 7  
>>> d  
?
```



# 수 다루기 - 정수

## ❖ 10진법, 2진법, 16진법으로 수 표현하기

10진수	2진수	16진수
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



# 수 다루기 - 정수

## ❖ 실습 7 (10진수를 16진수 문자열로 변환)

```
>>> hex(0)
'0x0'
>>> hex(255)
'0xff'
>>> hex(12345)
'0x3039'
```



# 수 다루기 - 정수

## ❖ 실습 8 (16진수 데이터를 변수에 할당)

```
>>> a = 0xFF
```

```
>>> a
```

```
255
```

```
>>> b = 0x20
```

```
>>> b
```

```
32
```

```
>>> c = 0x0
```

```
>>> c
```

```
0
```



# 수 다루기 - 정수

## ❖ 0x, 0b, 0o

- 16진수 접두사 : 0x (heXadecimal number)
- 2진수 접두사 : 0b(Binary number)
- 8진수 접두사 : 0o(Octal number)

## ❖ 실습 9 (10진수를 2진수 문자열로 변환)

```
>>> bin(0)
'0b0'
>>> bin(8)
'0b1000'
>>> bin(32)
'0b100000'
>>> bin(255)
'0b11111111'
```



# 수 다루기 - 정수

## ❖ 실습 10 (2진수를 변수에 할당)

```
>>> a = 0b100
>>> a
4
>>> b = 0b1001
>>> b
9
>>> c = 0b11111111
>>> c
255
```



# 수 다루기 - 정수

- ❖ 실습 11 (10진수를 8진수로 출력, 8진수를 변수에 할당)

```
>>> oct(8)
'0o10'
>>> oct(10)
'0o12'
>>> oct(64)
'0o100'
>>> a = 0o10
>>> a
8
>>> b = 0o12
>>> b
10
>>> c = 0o100
>>> c
64
```





# 수 다루기 - 실수

## ❖ 파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공

- “부동”은 뜰 부(浮), 움직일 동(動), 즉 떠서 움직인다는 뜻
- 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형

## ❖ 부동 소수형의 특징

- 부동 소수형은 8바이트만을 이용해서 수를 표현한다. 즉, 한정된 범위의 수만 표현할 수 있다.
- 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계를 갖는다.



# 수 다루기 - 실수

22/7의 결과는 무리수지만  
부동소수형은 소수점 이하  
15자리만 표현

## ❖ 실습 1

```
>>> b = 22 / 7
>>> b
3.142857142857143
>>> type(b)
<class 'float'>
```



# 수 다루기 - 실수

## ❖ 실습 2 (부동소수형의 사칙 연산)

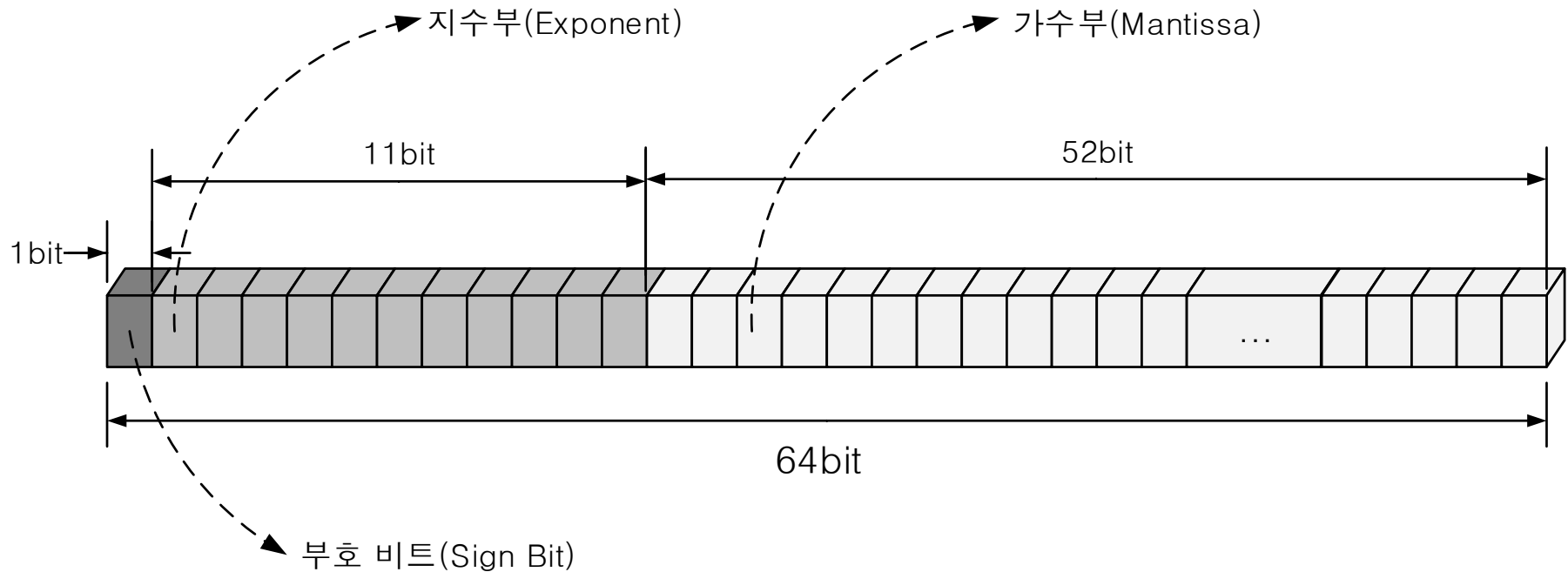
```
>>> a = 1.23 + 0.32
>>> a
1.55
>>> b = 3.0 - 1.5
>>> b
1.5
>>> c = 2.1 * 2.0
>>> c
4.2
>>> d = 4.5 // 2.0
>>> d
2.0
>>> e = 4.5 % 2.0
>>> e
0.5
>>> f = 4.5 / 2.0
>>> f
2.25
```



# 수 다루기 - 실수

## ❖ 부동 소수형의 구조

- IEEE 754



# 수 다루기 - 복소수

❖ 복소수는  $a \pm bi$ 의 꼴로 나타낼 수 있는 수.

- $a$ 와  $b$ 는 실수,  $i$ 는 허수 단위로  $i^2 = -1$ 을 만족.
- 켤레 복소수는 복소수 중 허수 부분의 부호가 반대인 복소수.
- 실수가 정수를 포함하는 것처럼 복소수도 실수를 포함
- 파이썬에서는 허수 단위를 나타내는 부호로  $i$  대신  $j$ 를 사용

```
>>> a = 2 + 3j
>>> a
(2+3j)
>>> type(a)
<class 'complex'>
>>> a.real
2.0
>>> a.imag
3.0
>>> a.conjugate() #켤레
(2-3j)
```



# 수 다루기 – math 모듈을 이용한 계산

## ❖ math 모듈을 사용하기 위한 import문

```
>>> import math
```

## ❖ $\pi$ 와 e

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

## ❖ 파이썬 코드에서 “.” 은 “~의” 로 해석 //구조체 생각해볼 것

- math.pi는 math(모듈)의 pi
- math.e는 math(모듈)의 e



# 수 다루기 – math 모듈을 이용한 계산

## ❖ 절대값, 버림과 반올림

함수	설명	비고
abs()	절대값 계산 함수	내장 함수
round()	반올림 계산 함수	내장 함수
trunc()	버림 계산 함수	math 모듈

## ❖ 실습 1 (abs() 함수)

```
>>> abs(10)
10
>>> abs(-10)
10
```

## ❖ 실습 2 (round() 함수)

```
>>> round(1.4)
1
>>> round(1.5)
2
```



# 수 다루기 – math 모듈을 이용한 계산

## ❖ 실습 3 (trunc() 함수)

```
>>> import math
>>> math.trunc(1.4)
1
>>> math.trunc(1.5)
1
>>> math.trunc(1.9)
1
```





# 수 다루기 – math 모듈을 이용한 계산

## ❖ 팩토리얼

- 팩토리얼은 1부터 어떤 양의 정수  $n$ 까지의 정수의 곱
  - 예)  $5! = 5 \times 4 \times 3 \times 2 \times 1$

함수	설명
factorial()	팩토리얼 계산 함수

## ❖ 실습 4 (factorial() 함수)

```
>>> math.factorial(1)
1
>>> math.factorial(5)
120
>>> math.factorial(10)
3628800
>>> math.factorial(100)
933262154439441526816
992388562667004907159
682643816214685929638
952175999932299156089
414639761565182862536
979208272237582511852
1091686400000000000000
000000000000
```



# 수 다루기 – math 모듈을 이용한 계산

## ❖ 제공과 제공근

함수	설명	비고
<b>**</b>	제공 연산	연산자
<b>pow()</b>	**와 같습니다.	math 모듈
<b>sqrt()</b>	제공근 연산	math 모듈

## ❖ 실습 5 (\*\*, pow() 함수)

```
>>> 3 ** 3
27
>>> import math
>>> math.pow(3,3)
27.0
```

## ❖ 실습 6 (sqrt() 함수)

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.sqrt(81)
9.0
```

## ❖ 실습 7 (세제곱근, 네제곱근)

```
>>> 27 ** (1/3)
3.0
>>> import math
>>> math.pow(81, 0.5)
9.0
```

# 수 다루기 – math 모듈을 이용한 계산

## ❖ 실습 5 (\*\*, pow()) 함수)

```
>>> 3 ** 3
27
>>> import math
>>> math.pow(3,3)
27.0
```

## ❖ 실습 6 (sqrt()) 함수)

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.sqrt(81)
9.0
```

## ❖ 실습 7 (세제곱근, 네제곱근)

```
>>> 27 ** (1/3)
3.0
>>> import math
>>> math.pow(81, 0.5)
9.0
```



# 수 다루기 – math 모듈을 이용한 계산

```
>>> import math
```

```
>>> math.log(4, 2)
```

```
2.0
```

```
>>> math.log(math.e)
```

```
1.0
```

```
>>> math.log(1000, 10)
```

```
2.9999999999999996
```

```
>>> math.log10(1000)
```

```
3.0
```

$\log_2 4$ 를 계산합니다.

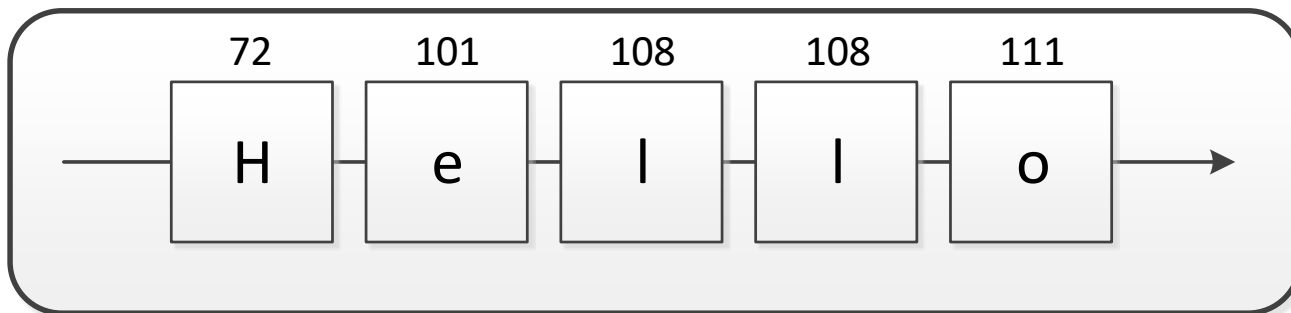
두 번째 매개 변수를 생략하면 자연 로그를 계산합니다.  $\ln(e)$ 와 같습니다.

$\text{math.log}(1000, 10)$ 과  $\text{math.log10}(1000)$ 은 모두  $\log_{10} 1000$ 을 계산합니다. 이 예제는 10을 밑으로 하는 수를 계산하는 데에는  $\log_{10}()$  함수가 적합함을 볼 수 있습니다.



# 텍스트 다루기

- ❖ 프로그래밍 언어마다 방식이 조금씩 다르긴 하지만 대부분은 다음 그림과 같이 개별 문자를 나타내는 수를 이어서 텍스트를 표현



- ❖ 파이썬에서는 텍스트를 다루는 자료형으로 **string**을 제공
  - string은 영어로 끈, 줄 등의 뜻을 갖고 있으므로 문자를 끈으로 가지런히 묶어놓은 것이라고 이해
  - 우리 말로는 문자열 → 문자열(文字列)도 문자를 가지런히 늘어놨다는 뜻



## 테스트 다르기

```
>>> a = 'Hello, World.'
>>> a
'Hello, World.'
>>> b = "안녕하세요."
>>> b
'안녕하세요.'
>>> c = '''어서와
파이썬은
처음이지?'''
>>> c
'어서와\n파이썬은\n처음이지?'
>>> d = """Welcome to
Python."""
>>> d
'Welcome to\nPython.'
>>> type(d)
<class 'str'>
```



# 텍스트 다루기

## ❖ 문자열을 다룰 때의 + 연산자는 두 문자열을 하나로 이어 붙임

- + 연산자가 문자열을 결합한다고 해서 - 연산자가 문자열을 분리하는 것은 아님

## ❖ 실습 2 (+ 연산자)

```
>>> hello = 'Hello'
>>> world = 'World'
>>> hello_world = hello + ', ' + world
>>> hello_world
'Hello, World'
```

## ❖ 문자열 분리(슬라이싱 Slicing)는 [와 ] 연산자를 통해 수행함

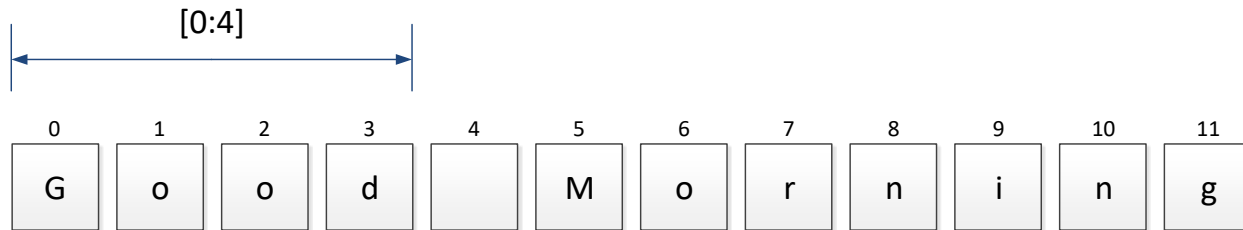


# 텍스트 다루기

- ❖ 문자열 분리(슬라이싱 Slicing)는 [와 ] 연산자를 통해 수행함
- ❖ 실습 3 (문자열 분리)

```
>>> s = 'Good Morning'  
>>> s[0:4]  
'Good'
```

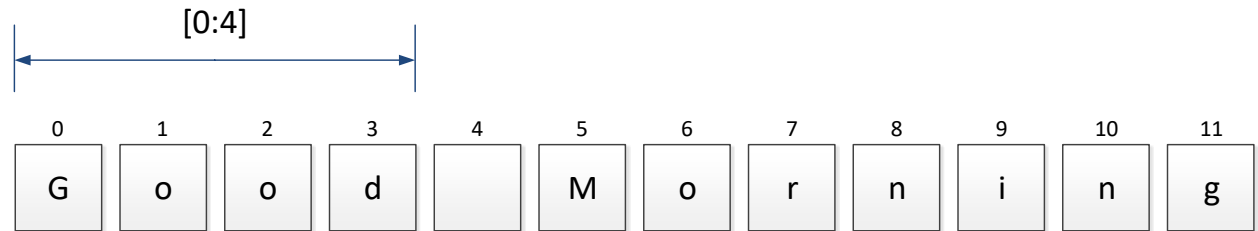
문자열 s의 0번째 문자부터 4번째 문자 앞까지를 분리합니다.





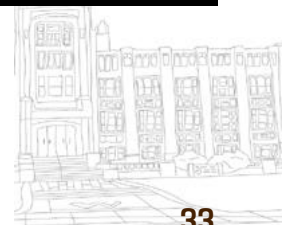
# 텍스트 다루기

- ❖ 슬라이싱은 문자열 뿐 아니라 다른 순서열 자료형에서도 사용 가능
- ❖ 문자열이든 순서열이든 슬라이싱을 하더라도 원본은 그대로 유지
- ❖ 실습 4 (문자열 분리2)



```
>>> a = 'Good Morning'
>>> b = a[0:4]
>>> c = a[5:12]
>>> a
'Good Morning'
>>> b
'Good'
>>> c
'Morning'
```

문자열 a를 슬라이싱해서 b를 만들어내도 a는 여전히 'Good Morning'입니다. 따라서 c를 만들어 낼 때도 a의 원본 그대로가 사용됩니다.



# 텍스트 다루기

- ❖ 특정 위치에 있는 문자를 참조하고 싶을 때는 대괄호 [와 ] 사이에 사이에 첨자(Index) 번호 하나만 입력

- ❖ 실습 5

```
>>> a = 'Good Morning'
>>> a[0]
'G'
>>> a[8]
'n'
```

- ❖ in 연산자는 프로그래머가 원하는 부분이 문자열 안에 존재하는지를 확인

- ❖ 실습 6

```
>>> a = 'Good Morning'
>>> 'Good' in a
True
>>> 'X' in a
False
>>> 'Evening' in a
False
```



# 텍스트 다루기

- ❖ `len()` : 순서열 길이를 재는 함수. 문자열에도 사용 가능.
- ❖ 실습 7 (`len()`)

```
>>> a = 'Good Morning'  
>>> len(a)  
12
```



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>startswith()</b>	<p>원본 문자열이 매개변수로 입력한 문자열로 시작되는지를 판단합니다. 결과는 True 또는 False로 나옵니다.</p> <pre>&gt;&gt;&gt; a = 'Hello' &gt;&gt;&gt; a.startswith('He') True &gt;&gt;&gt; a.startswith('lo') False &gt;&gt;&gt;</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>endswith()</b>	<p>원본 문자열이 매개변수로 입력한 문자열로 끝나는지를 판단합니다. 결과는 True 또는 False로 나옵니다.</p> <pre>&gt;&gt;&gt; a = 'Hello' &gt;&gt;&gt; a.endswith('He') False &gt;&gt;&gt; a.endswith('lo') True</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>find()</b>	<p>원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 앞에서부터 찾습니다. 존재하지 않으면 -1을 결과로 내놓습니다.</p> <pre>&gt;&gt;&gt; a = 'Hello' &gt;&gt;&gt; a.find('l') 2 &gt;&gt;&gt; a.find('H') 0 &gt;&gt;&gt; a.find('K') -1</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>rfind()</b>	<p>원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 뒤에서부터 찾습니다. 존재하지 않으면 -1을 결과로 내놓습니다.</p> <pre>&gt;&gt;&gt; a = 'Hello' &gt;&gt;&gt; a.rfind('H') 0 &gt;&gt;&gt; a.rfind('lo') 3 &gt;&gt;&gt; a.rfind('M') -1</pre>



# 텍스트 다루기 - 문자열 메소드

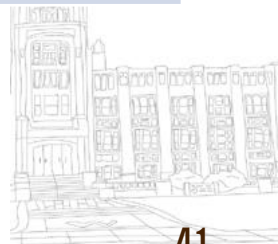
메소드	설명
<b>count()</b>	<p>원본 문자열 안에 매개변수로 입력한 문자열이 몇 번 등장하는지를 셉니다.</p> <pre>&gt;&gt;&gt; a = 'Hello' &gt;&gt;&gt; a.count('l') 2</pre>





# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>lstrip()</b>	원본 문자열 왼쪽에 있는 공백을 제거합니다.  >>> '    Left Strip'.lstrip() 'Left Strip'
<b>rstrip()</b>	원본 문자열 오른쪽에 있는 공백을 제거합니다.  >>> 'Right Strip    '.rstrip() 'Right Strip'
<b>strip()</b>	원본 문자열 양쪽에 있는 공백을 제거합니다.  >>> '    Strip    '.strip() 'Strip'



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>isalpha()</b>	<p>원본 문자열이 숫자와 기호를 제외한 알파벳(영문, 한글 등)으로만 이루어져 있는지를 평가합니다.</p> <pre>&gt;&gt;&gt; 'ABCDefgh'.isalpha() True &gt;&gt;&gt; '123ABC'.isalpha() False</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>isnumeric()</b>	원본 문자열이 수로만 이루어져 있는지를 평가합니다. . >>> '1234'.isnumeric() True >>> '123ABC'.isnumeric() False



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>isalnum()</b>	<p>원본 문자열이 알파벳과 수로만 이루어져 있는지를 평가합니다.</p> <pre>&gt;&gt;&gt; '1234ABC'.isalnum() True &gt;&gt;&gt; '1234'.isalnum() True &gt;&gt;&gt; 'ABC'.isalnum() True &gt;&gt;&gt; '1234 ABC'.isalnum() False</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>replace()</b>	<p>원본 문자열에서 찾고자 하는 문자열을 바꾸고자 하는 문자열로 변경합니다.</p> <pre>&gt;&gt;&gt; a = 'Hello, World' &gt;&gt;&gt; b = a.replace('World', 'Korea') &gt;&gt;&gt; a 'Hello, World' &gt;&gt;&gt; b 'Hello, Korea'</pre>
<b>split()</b>	<p>매개변수로 입력한 문자열을 기준으로 원본 문자열을 나누어 리스트를 만듭니다. 리스트는 목록을 다루는 자료형이며 다음 장에서 자세히 다룹니다.</p> <pre>&gt;&gt;&gt; a = 'Apple, Orange, Kiwi' &gt;&gt;&gt; b = a.split(',') &gt;&gt;&gt; b ['Apple', ' Orange', ' Kiwi'] &gt;&gt;&gt; type(b) &lt;class 'list'&gt;</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>upper()</b>	<p>원본 문자열을 모두 대문자로 바꾼 문자열을 내놓습니다.</p> <pre>&gt;&gt;&gt; a = 'lower case' &gt;&gt;&gt; b = a.upper() &gt;&gt;&gt; a 'lower case' &gt;&gt;&gt; b 'LOWER CASE'</pre>
<b>lower()</b>	<p>원본 문자열을 모두 소문자로 바꾼 문자열을 내놓습니다.</p> <pre>&gt;&gt;&gt; a = 'UPPER CASE' &gt;&gt;&gt; b = a.lower() &gt;&gt;&gt; a 'UPPER CASE' &gt;&gt;&gt; b 'upper case'</pre>



# 텍스트 다루기 - 문자열 메소드

메소드	설명
<b>format()</b>	<p>형식을 갖춘 문자열을 만들 때 사용합니다. 문자열 안에 중괄호 {와 }로 다른 데이터가 들어갈 자리를 만들어 두고 format() 함수를 호출할 때 이 자리에 들어갈 데이터를 순서대로 넣어주면 원하는 형식의 문자열을 만들어 낼 수 있습니다.</p> <pre>&gt;&gt;&gt; a = 'My name is {0}. I am {1} years old.'.format('Mario', 40) &gt;&gt;&gt; a 'My name is Mario. I am 40 years old.' &gt;&gt;&gt; b = 'My name is {name}. I am {age} years old.'.format(name='Luigi', age=35) &gt;&gt;&gt; b 'My name is Luigi. I am 35 years old.'</pre>



# 수에서 텍스트로, 텍스트에서 수로

## ❖ 이 코드는 '정상적으로' 동작할까?

```
a = input()
b = input()
result = a * b
```

a와 b는 문자열입니다. 이 코드는 파이썬이 수행할 수 없습니다.

- input() 함수의 결과는 문자열이므로 \* 연산자를 사용할 수 없음.

## ❖ 문자열을 숫자로 바꾸기 위해서는 int(), float(), complex()를 사용

```
>>> int('1234567890')
1234567890
>>> float('123.4567')
123.4567
>>> complex('1+2j')
(1+2j)
```





# 수에서 텍스트로, 텍스트에서 수로

## ❖ 실습 2 : *04/input\_multiply.py*

```
a = input("첫 번째 수를 입력하세요. : ")  
b = input("두 번째 수를 입력하세요. : ")  
  
result = int(a) * int(b)  
  
print("{0} * {1} = {2}".format(a, b, result))
```



# 수에서 텍스트로, 텍스트에서 수로

- ❖ 숫자를 문자열로 바꾸기 위해서는 `str()`을 사용
- ❖ 실습 3

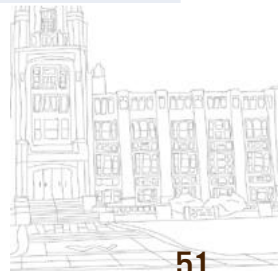
```
>>> import math
>>> type(math.pi)
<class 'float'>
>>> text = "원주율은 " + str(math.pi) + "입니다."
>>> text
'원주율은 3.141592653589793입니다.'
```



# 비트 다루기

## ❖ 파이썬에서 제공하는 비트 연산자

연산자	이름	설명
<<	왼쪽 시프트 연산자	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 왼쪽으로 이동시킵니다.
>>	오른쪽 시프트 연산자	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 오른쪽으로 이동시킵니다.
&	논리곱 (AND) 연산자	두 피연산자의 비트 논리곱을 수행합니다.
	논리합(OR) 연산자	두 피연산자의 비트 논리합을 수행합니다.
^	배타적 논리합(XOR) 연산자	두 피연산자의 비트 배타적 논리합을 수행합니다.
~	보수(NOT) 연산자	피연산자의 비트를 0은 1로, 1은 0으로 반전시킵니다. 단항 연산자입니다.



# 비트 다루기 – 시프트 연산자

❖ 시프트 연산자(Shift Operator)는 비트를 왼쪽이나 오른쪽으로 이동시키는 기능을 수행

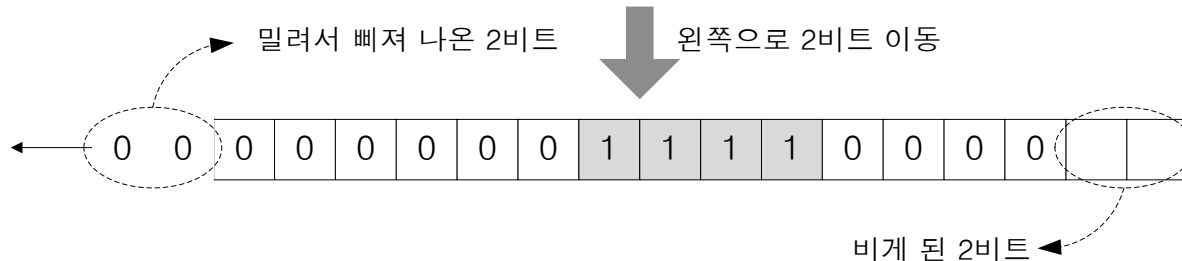
❖ 왼쪽 시프트 연산

- 10진수 240(16개의 비트로 표현)

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- 이 비트를 전체적으로 왼쪽으로 2비트 이동하면 다음과 같음.

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



- 비어있는 비트에 0을 채우면 왼쪽 시프트 연산 완료

0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



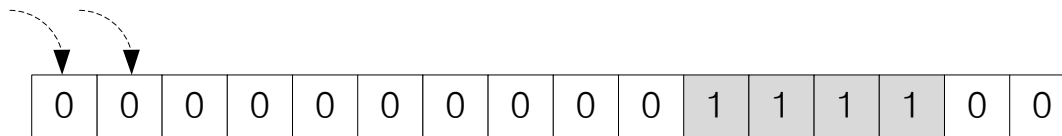
# 비트 다루기 – 시프트 연산자

## ❖ 오른쪽 시프트 연산(양수)

- 10진수 240(16개의 비트로 표현)를 오른쪽으로 2비트 이동



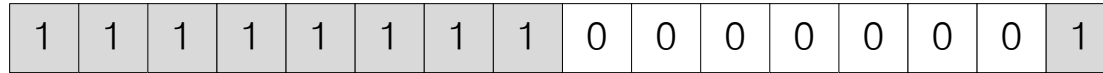
- 이동하고 남은 비트에 0을 채우면 오른쪽 시프트 연산 완료



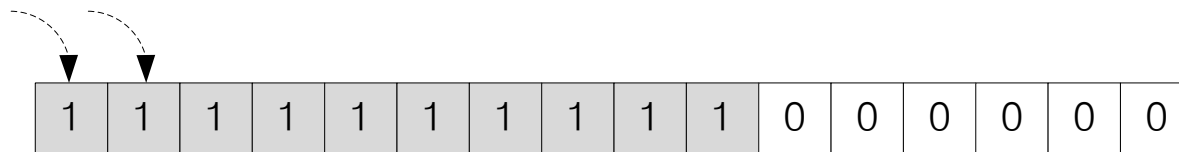
# 비트 다루기 – 시프트 연산자

## ❖ 오른쪽 시프트 연산(음수)

- 10진수 -255(16개의 비트로 표현)를 오른쪽으로 2비트 이동



- 이동하고 남은 비트에 1을 채우면 오른쪽 시프트 연산 완료



# 비트 다루기 – 시프트 연산자

## ❖ 실습 1

```
>>> a = 240 # 00000000 11110000
>>> a
240
>>> a << 2 # 00000011 11000000
960
>>> a >> 2 # 00000000 00111100
60
```

원본 데이터

움길 비트의 수

## ❖ 실습 2

```
>>> a = 1
>>> hex(a)
0x1
>>> hex(a << 1)
0x2
>>> hex(a << 2)
0x4
>>> hex(a << 5)
0x20
```

```
>>> b = 255
>>> hex(b)
0xff
>>> hex(b >> 1)
0x7f
>>> hex(b >> 2)
0x3f
>>> hex(b >> 5)
0x7
```

```
>>> c = -255
>>> hex(c)
-0xff
>>> hex(c >> 1)
-0x80
>>> hex(c >> 2)
-0x40
>>> hex(c >> 5)
-0x8
```



# 비트 다루기 – 비트 논리 연산자

- ❖ 논리 연산은 참 또는 거짓의 진리 값을 피연산자로 하는 연산
- ❖ 비트 논리 연산(Bitwise Logical Operation)도 데이터의 각 비트에 대해 수행하는 논리 연산
- ❖ 파이썬에서 제공하는 비트 논리 연산자

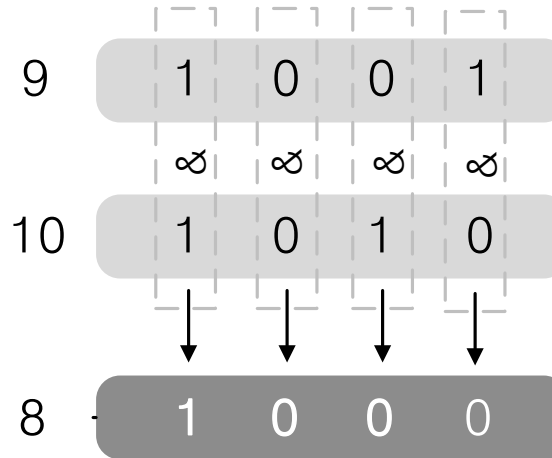
연산자	이름	설명
&	논리곱(AND) 연산자	두 피연산자의 비트에 대해 논리곱을 수행합니다.
	논리합(OR) 연산자	두 피연산자의 비트에 대해 논리합을 수행합니다.
^	배타적 논리합(XOR) 연산자	두 피연산자의 비트의 대해 배타적 논리합을 수행합니다.
~	보수(NOT) 연산자	피연산자의 비트에 대해 0은 1로, 1은 0으로 반전시킵니다. 단항 연산자입니다.





# 비트 다루기 – 비트 논리 연산자

- ❖ 논리곱: 두 비트 모두가 1(참)이어야 결과도 1(참)
- ❖ 논리곱 연산자는 &

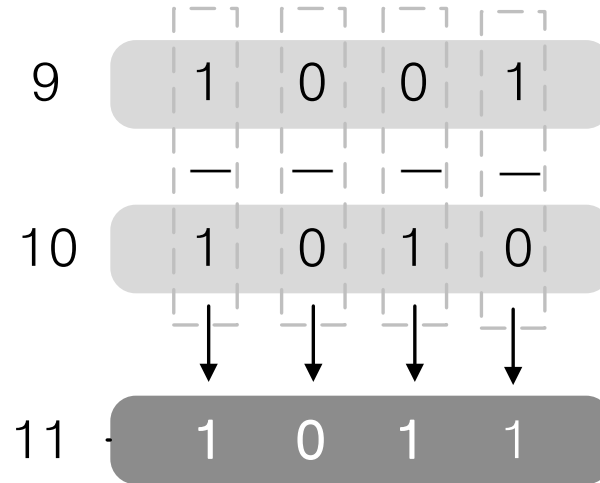


```
>>> 9 & 10  
8
```



# 비트 다루기 – 비트 논리 연산자

- ❖ 논리합: 둘 중 하나라도 참(1)이면 결과도 참(1)
- ❖ 논리합 연산자는 |



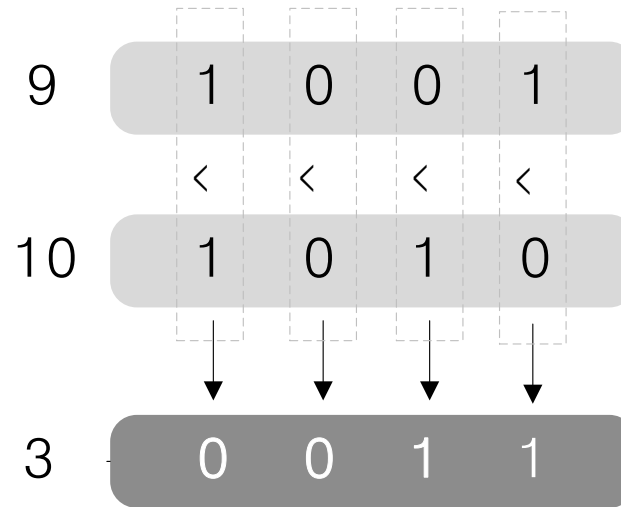
```
>>> 9 | 10  
11
```

- ❖ 실습 2



# 비트 다루기 – 비트 논리 연산자

- ❖ 배타적 논리합: 두 피연산자의 진리값이 서로 달라야 참(1)
- ❖ 배타적 논리합 연산자는 ^



```
>>> 9 ^ 10  
3
```

- ❖ 실습 3



# 비트 다루기 – 비트 논리 연산자

- ❖ 보수 연산: 피연산자의 비트를 0에서 1로, 1에서 0으로 뒤집음.
- ❖ 보수 연산자는 ~

```
>>> a = 255  
>>> ~a  
-256
```

255

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

보수 연산

-256

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- ❖ 실습 4





# Thank You !

뇌를 자극하는 파이썬3