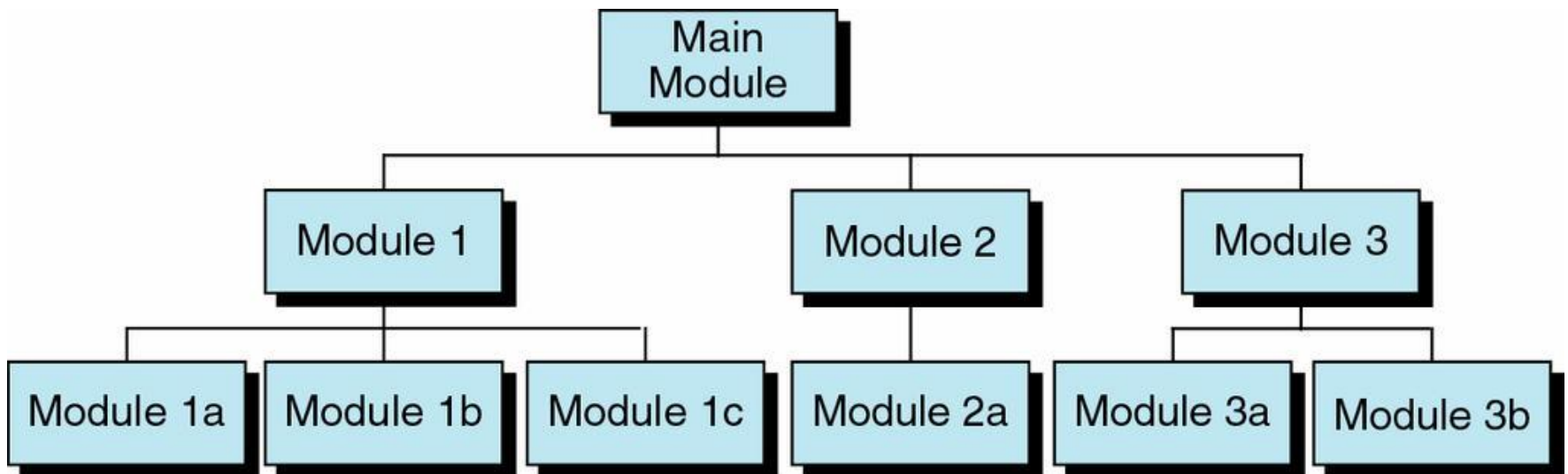


# 변수와 데이터형

# 함수

## ❖ 모듈

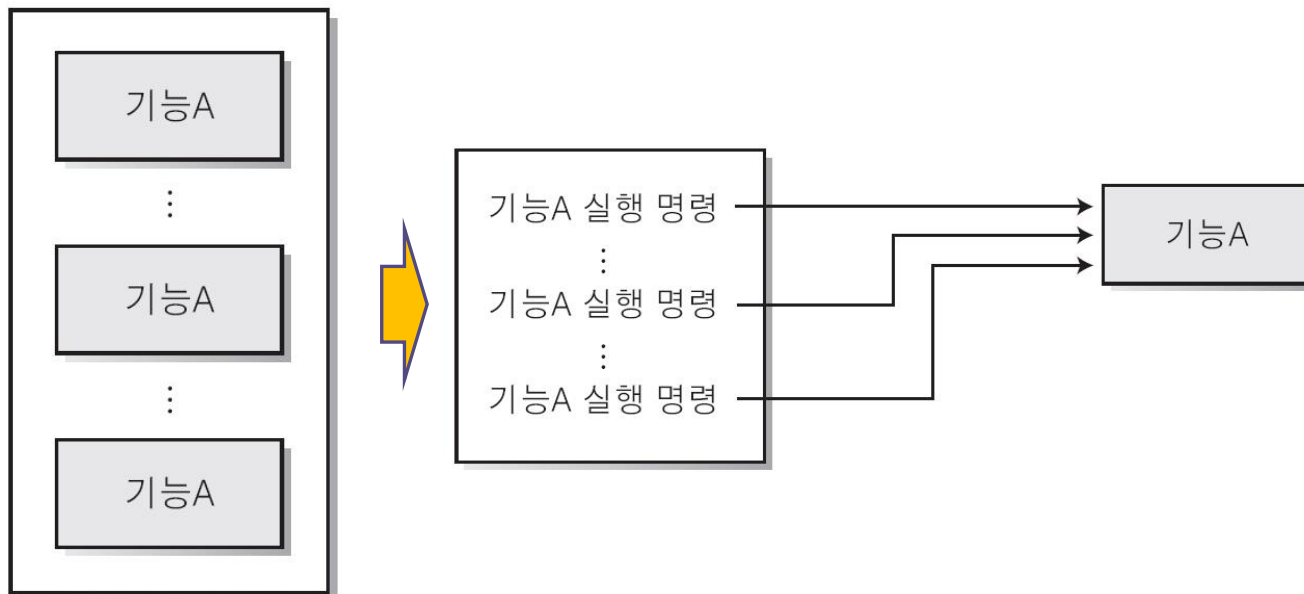
- 독립되어 있는 프로그램의 일부분
- 모듈러 프로그래밍 : 모듈 개념을 사용하는 프로그래밍 기법
- 모듈러 프로그래밍의 장점
  - ✓ 각 모듈들은 독자적으로 개발 가능
  - ✓ 다른 모듈과 독립적으로 변경 가능
  - ✓ 유지 보수가 쉬움
  - ✓ 모듈의 재사용 가능



# 함수

## ❖ 함수(function)

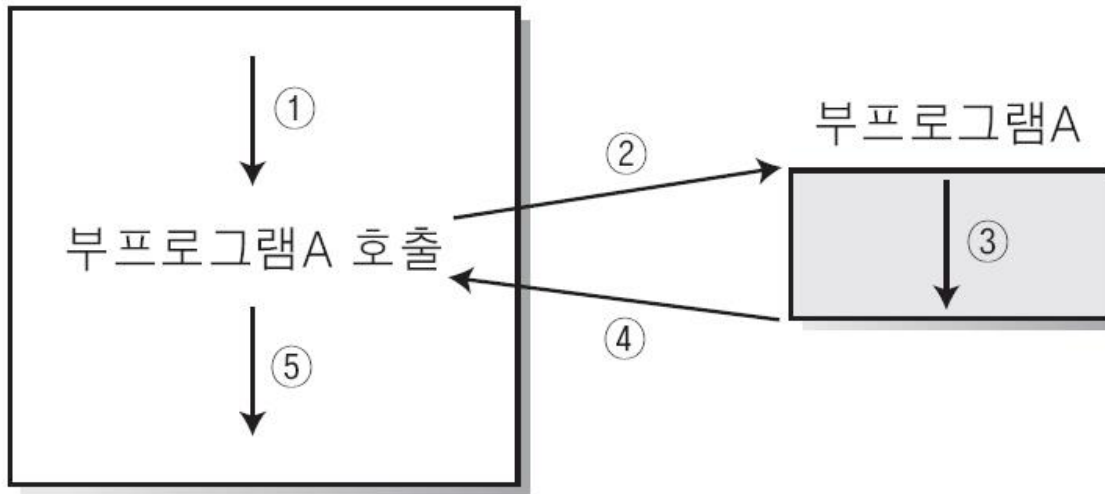
- 특정한 작업을 수행하는 독립적인 부분으로 명령어(코드)들의 그룹
- 함수 호출(function call): 함수를 호출하여 사용하는 것
- 서로 다른 장소에서 여러번 반복적으로 실행되는 명령어들을 함수 형태로 정의 호출하여 반복적인 프로그래밍을 피함
- 함수 종류
  - ✓ 라이브러리 함수 : 여러 프로그램에서 호출하여 사용할 수 있도록 미리 정의
  - ✓ 사용자 정의 함수 : 모듈화된 프로그램을 작성하기 위해 사용자가 직접 작성



# 함수

## ❖ 함수 호출

- 함수는 함수 이름을 통하여 호출
- 함수가 호출되면 함수 내에 정의된 문장들이 실행
- 함수 내 문장들이 모두 실행된 후에는 그 함수를 호출한 곳으로 값을 반환
- 함수 호출문 다음 문장의 실행이 재개



# print() 함수를 사용한 다양한 출력

## ❖ print() 함수

- 출력 장치인 모니터에 결과물을 출력하기 위한 라이브러리 함수
- print( ) 함수 서식 예

```
print("안녕하세요?")
```

결과는 '안녕하세요?' 이다

```
❶ print("100")
```

```
❷ print("%d" % 100)
```

- ❶의 결과로 나온 100은 숫자 100(백)이 아닌 문자 100(일명영)이다,  
" " 안의 내용이 문자든 숫자든 무조건 문자로 취급한다.  
❷의 결과로 나온 100은 숫자 100(백)을 의미한다

```
❸ print("100 + 100")
```

```
❹ print("%d" % (100 + 100))
```

- ❸은 100+100이 출력되고,  
❹는 숫자 100과 숫자 100을 더한 결과인 숫자 200을 출력한다.

```
❺ print("%d" % (100, 200))
```

```
❻ print("%d %d" % (100))
```

- ❺는 %d가 하나밖에 없는데 숫자가 2개이고,  
❻은 %d가 2개인데 숫자는 하나라 서로 짝이 맞지 않다.  
❻은 단순히 %d를 하나 삭제하면 되지만 ❺는 숫자 2개를 출력하려면 %d가 2개 필요하므로 수정한다.

```
print( "%d %d" % ( 100 , 200 ) )
```

# print() 함수를 사용한 다양한 출력

❖ print( ) 함수에 사용할 수 있는 서식

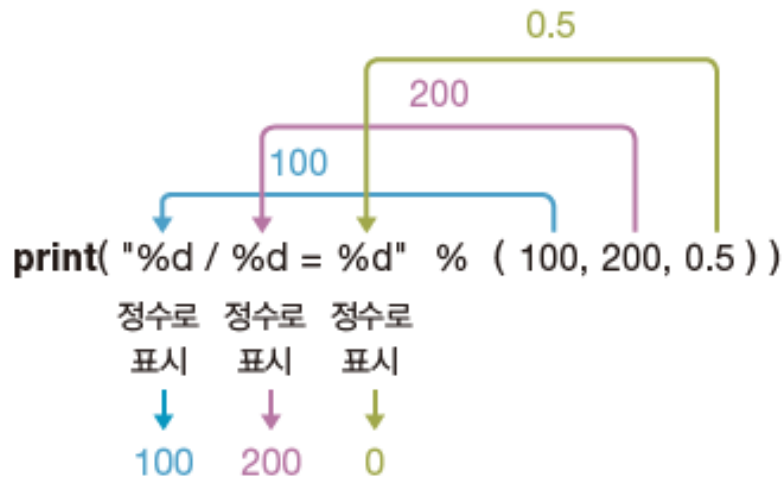
서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14	실수(소수점이 붙은 수)
%c	"b", "한"	한 글자
%s	"안녕", "abcdefg", "a"	두 글자 이상인 문자열

# print() 함수를 사용한 다양한 출력

## ❖ print( ) 함수를 사용한 다양한 출력

```
print("%d / %d = %d" % (100, 200, 0.5))
```

결과는 100/200=0이다,



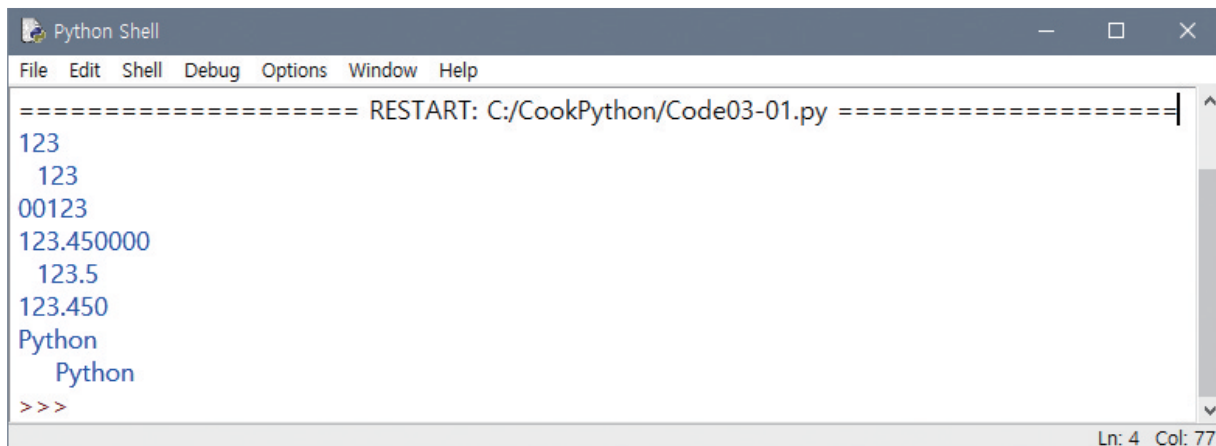
## ➤ 코드를 실수가 출력될 수 있도록 수정

```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```

# print() 함수를 사용한 다양한 출력

## ❖ print( ) 함수를 사용한 출력 예

```
1 print("%d" % 123)
2 print("%5d" % 123)
3 print("%05d" % 123)
4
5 print("%f" % 123.45)
6 print("%7.1f" % 123.45)
7 print("%7.3f" % 123.45)
8
9 print("%s" % "Python")
10 print("%10s" % "Python")
```



The screenshot shows a Python Shell window with the following output:

```
===== RESTART: C:/CookPython/Code03-01.py =====
123
  123
00123
123.450000
  123.5
123.450
Python
  Python
>>>
```

The status bar at the bottom right indicates "Ln: 4 Col: 77".



# print() 함수를 사용한 다양한 출력

## ❖ print( ) 함수에서 정수형 데이터 서식 지정

"%d" →  숫자의 자릿수만큼 정렬

"%5d" →  오른쪽에 붙여서 정렬  
다섯 자리 확보


"%05d" →  오른쪽에 붙여서 정렬  
빈칸을 0으로 채움  
다섯 자리 확보

# print() 함수를 사용한 다양한 출력

## ❖ print( ) 함수에서 실수형 데이터 서식 지정

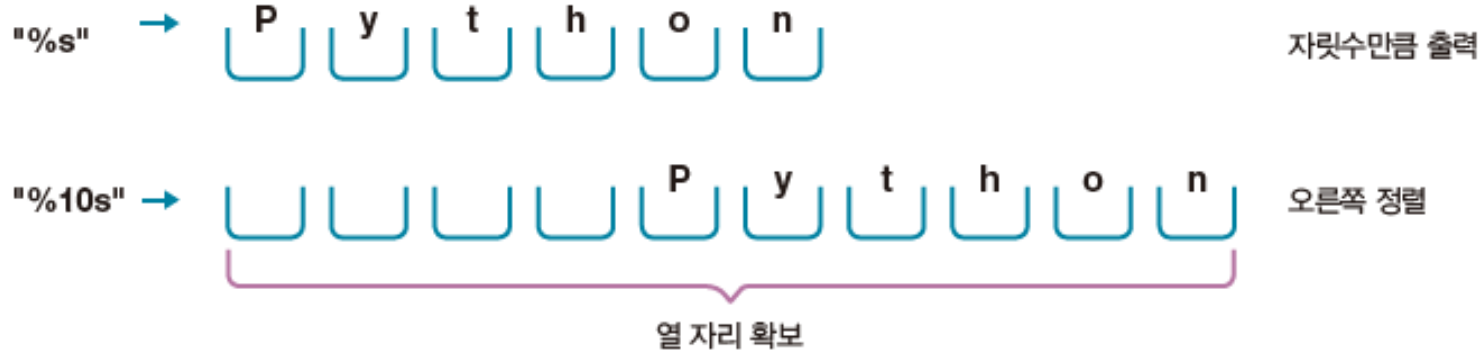
"%f" →  소수점 아래 여섯 자리까지  
무조건 출력

"%7.1f" →  일곱 자리 확보  
소수점 아래 첫째 자리만 출력  
소수점 아래 둘째 자리에서 반올림

"%7.3f" →  일곱 자리 확보  
소수점 아래 셋째 자리까지 출력  
오른쪽 빈칸은 0으로 채움

# print() 함수를 사용한 다양한 출력

## ❖ print( ) 함수에서 문자열 데이터 서식 지정

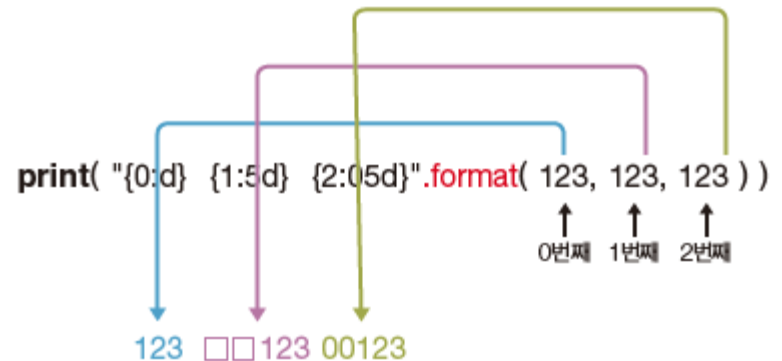


# print() 함수를 사용한 다양한 출력

## ❖ format( ) 함수를 이용한 서식 지정

- { }를 함께 사용해 출력 서식 지정
- %와 동일한 기능을 지원하며 괄호 내에 출력 형식을 지정

```
print("%d %5d %05d" % (123, 123, 123))  
print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))
```



- .format을 사용해 출력 순서 지정

```
print("{2:d} {1:d} {0:d}".format(100, 200, 300))
```

# print() 함수를 사용한 다양한 출력

## ❖ 이스케이프 문자

이스케이프 문자	역할	설명
\n	새로운 줄로 이동	<code>Enter</code> 를 누른 효과
\t	다음 탭으로 이동	<code>Tab</code> 을 누른 효과
\b	뒤로 한 칸 이동	<code>Backspace</code> 를 누른 효과
\\	\ 출력	
\'	' 출력	
\"	" 출력	

➤ 강제 행 넘기기는 'wn'을 사용

```
print("한 행입니다. 또 한 행입니다.")  
print("한 행입니다. \n또 한 행입니다.")
```

# print() 함수를 사용한 다양한 출력

## ❖ 이스케이프 문자 활용 예

```
1 print("\n줄바꿈\n연습 ")
2 print("\t탭키\t연습")
3 print("글자가 \"강조\"되는 효과1")
4 print("글자가 \'강조\'되는 효과2")
5 print("\\\\\\\\ 역슬래시 세 개 출력")
6 print(r"\n \t \" \"를 그대로 출력")
```



A screenshot of a Python Shell window titled "Python Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the output of a Python script. At the top, it says "==== RESTART: C:/CookPython/Code03-02.py". The output consists of several lines: a blank line followed by "줄바꿈" and "연습" on the next line; "탭키" followed by "연습" on the next line; "글자가 \"강조\"되는 효과1" on the next line; "글자가 \'강조\'되는 효과2" on the next line; "\\\\\\\\ 역슬래시 세 개 출력" on the next line; and a line with escaped characters: "Wn Wt W\" \"를 그대로 출력". The prompt ">>>" is visible at the bottom of the text area.

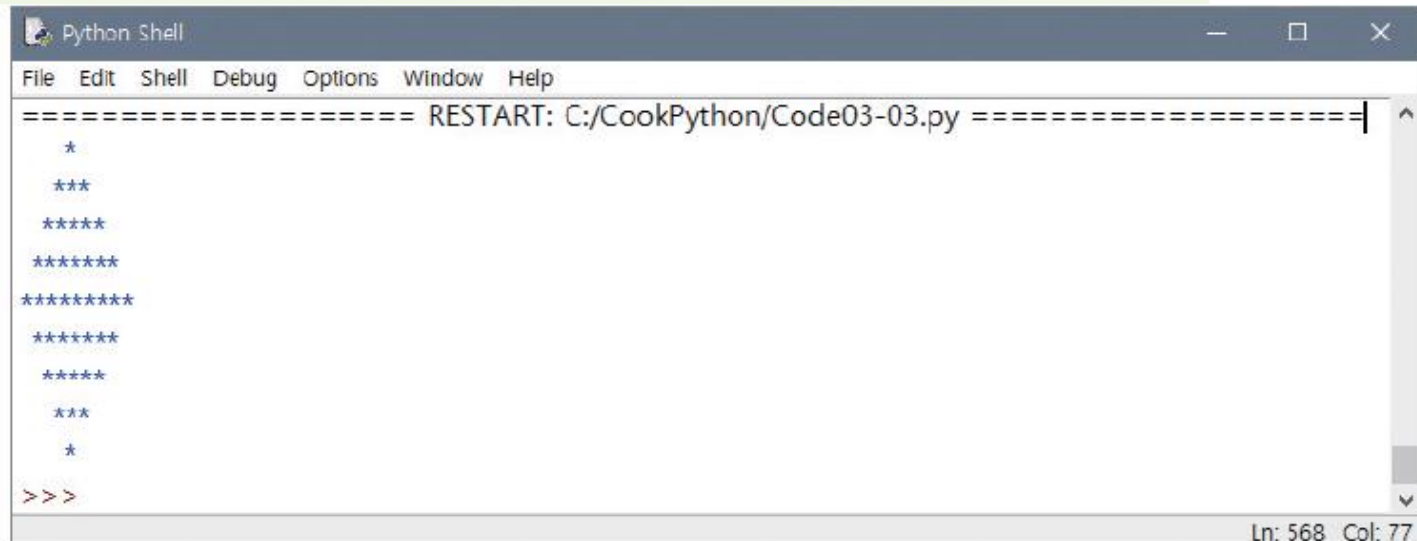
```
==== RESTART: C:/CookPython/Code03-02.py

줄바꿈
연습
    탭키    연습
글자가 "강조"되는 효과1
글자가 '강조'되는 효과2
\\\\\\\\ 역슬래시 세 개 출력
Wn Wt W" \"를 그대로 출력
>>>
```

# print() 함수를 사용한 다양한 출력

## ❖ 다이아몬드 모양 출력

```
1 print("  *  ")
2 print(" *** ")
3 print(" ***** ")
4 print(" ******* ")
5 print("*****")
6 print(" ******* ")
7 print(" ***** ")
8 print(" *** ")
9 print("  *  ")
```

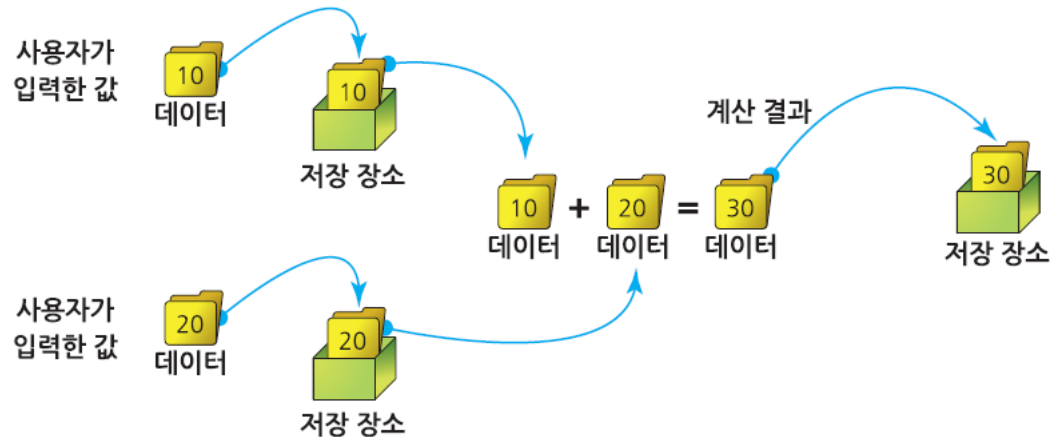


The screenshot shows a Python Shell window with the following menu: File, Edit, Shell, Debug, Options, Window, Help. The command prompt shows the execution of a script: `===== RESTART: C:/CookPython/Code03-03.py =====`. The output is a diamond shape made of asterisks, matching the code in the previous block. The status bar at the bottom right indicates `Ln: 568 Col: 77`.

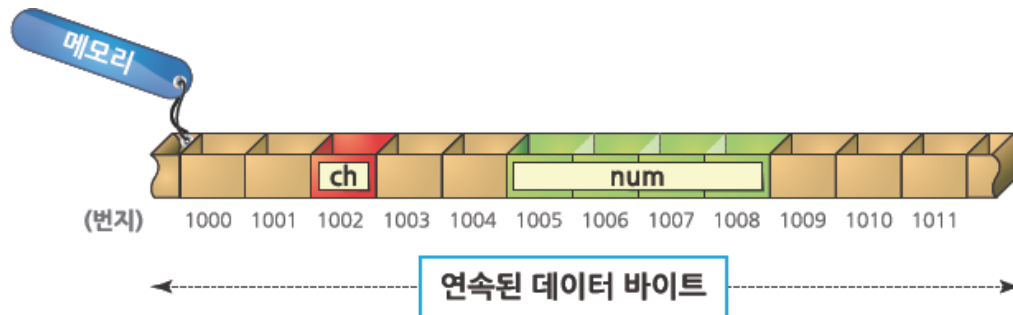
# 변수의 선언과 사용

## ❖ 변수

- 변수는 변경 가능한 데이터를 저장
  - ✓ 특정 값을 변수에 저장하거나 메모리에 보관된 변수의 값을 읽어올 수 있음



- 연속된 데이터 바이트의 모임으로 메모리의 각 바이트는 주소를 갖음
  - ✓ 변수는 물리적으로 기억장치인 메모리에 위치를 대신하는 고유한 이름





# 변수의 선언과 사용

## ❖ 변수의 선언

- 변수 선언은 데이터를 저장한 공간을 준비
- 파이썬은 C/C++, 자바 등과는 달리 변수를 선언하지 않아도 됨
  - ✓ 긴 코드를 작성할 때는 사용될 변수를 미리 계획적으로 준비하는 것이 더 효율적

```
boolVar = True  
intVar = 0  
floatVar = 0.0  
strVar = ""
```

**TIP** • 이 구문은 다음과 같이 표현해도 된다.

```
boolVar, intVar, floatVar, strVar = True, 0, 0.0, ""
```

- 가장 많이 사용하는 변수는 불형(Boolean, True 또는 False 저장), 정수형, 실수형, 문자열

# 변수의 선언과 사용

## ❖ 데이터형 확인

- 변수는 데이터형에 따라 저장 공간의 크기와 저장 방식이 결정
  - ✓ 불형(Boolean, True 또는 False 저장), 정수형, 실수형, 문자열
- type( ) 함수를 사용하면 변수의 데이터형을 확인
  - ✓ bool(불형), int(정수), float(실수), str(문자열)형으로 생성된 것을 확인

```
,  
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

### 출력 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

# 변수의 선언과 사용

## ❖ 변수명 규칙

X

- 대·소문자를 구분
  - ✓ myVar와 MyVar는 다른 변수
- 문자, 숫자, 언더바(\_)를 포함할 수 있으나 하지만 숫자로 시작하면 안 됨
  - ✓ var2(O), \_var(O), var\_2(O), 2Var(X)
- 예약어는 변수명으로 쓰면 안 됨
  - ✓ 파이썬의 예약어는 True, False, None, and, or, not, break, continue, return, if, else, elif, for, while, except, finally, gloval, import, try 등



+



# 변수의 선언과 사용

## ❖ 변수의 사용(1/5)

- 변수는 값을 담으면(대입하면) 사용 가능
- 변수에 있던 **기존 값은 없어지고 새로 입력한 값으로 변경**

```
boolVar = False  
intVar = 100  
floatVar = 123.45  
strVar = "안녕?"
```



# 변수의 선언과 사용

## ❖ 변수의 사용(2/5)

- 변수에는 변수의 **값**을 넣을 수도 있고, **계산 결과**를 넣을 수도 있음

```
var2 = 200  
var1 = var2
```



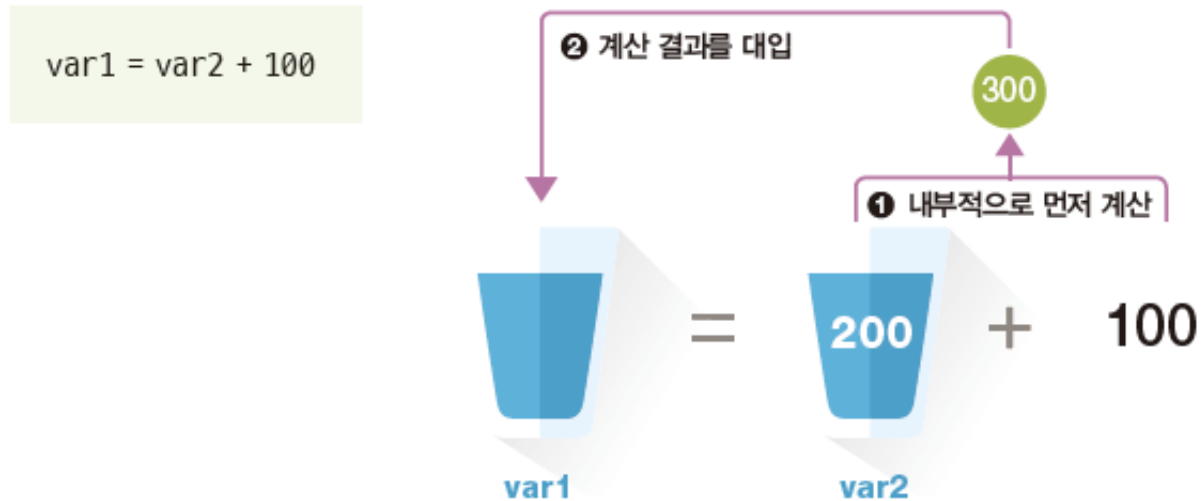
```
var1 = 100 + 100
```



# 변수의 선언과 사용

## ❖ 변수의 사용(3/5)

- 변수에는 숫자와 변수의 연산을 넣을 수도 있음



# 변수의 선언과 사용

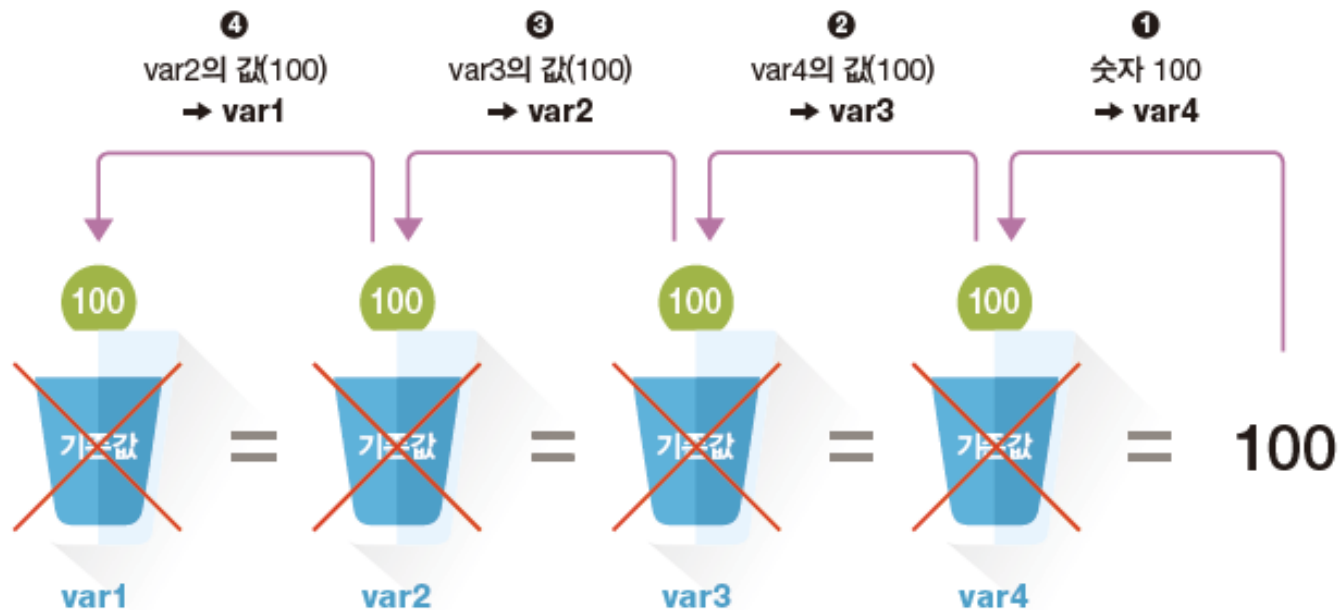
## ❖ 변수의 사용(4/5)

- 변수에 연속된 값을 대입하는 방식

```
var1 = var2 = var3 = var4 = 100
```

또는

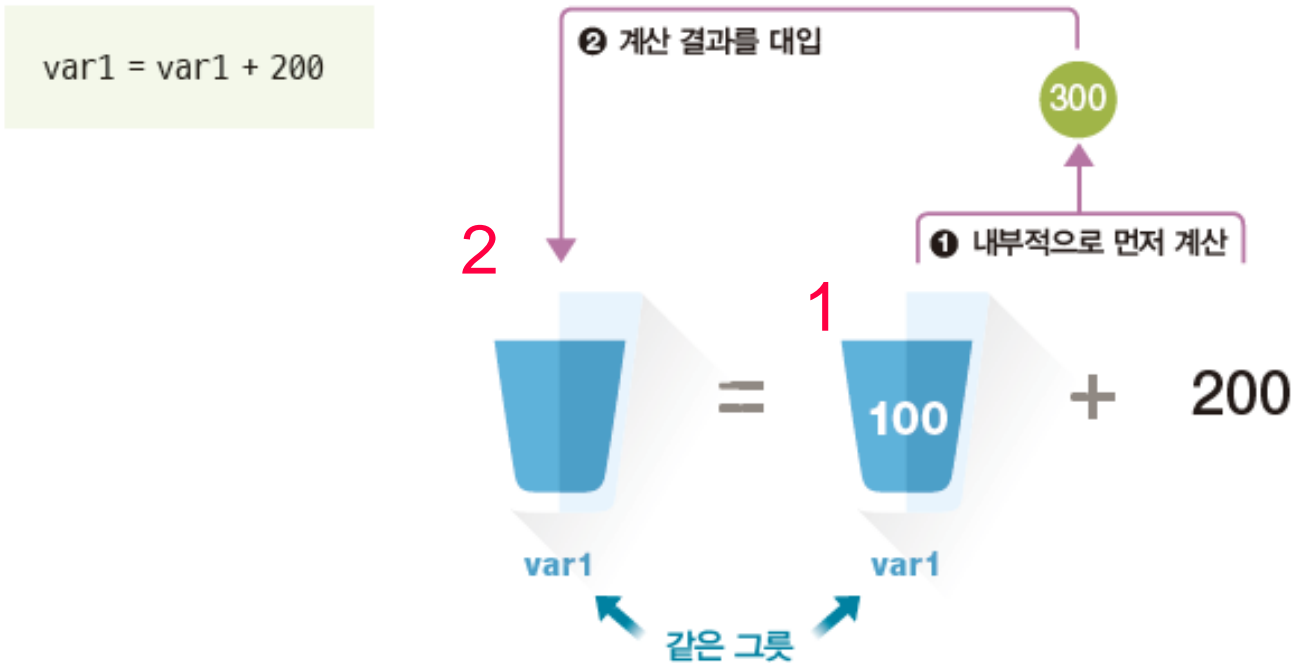
```
var4 = 100  
var3 = var4  
var2 = var3  
var1 = var2
```



# 변수의 선언과 사용

## ❖ 변수의 사용(5/5)

- 변수에 연산 결과를 자신의 값으로 다시 대입하는 방식





# 데이터 표현 단위와 진수 변환

## ❖ 비트

- 컴퓨터에서 표현할 수 있는 제일 작은 단위는 비트(Bit)
- 비트는 0과 1만 존재하므로 1비트로는 두 가지를 표현 가능

전기 스위치				
의미	꺼짐 , 꺼짐	꺼짐 , 켜짐	켜짐 , 꺼짐	켜짐 , 켜짐
2진수	00	01	10	11
10진수	0	1	2	3

$n$ 개의 전기 스위치로 표현할 수 있는 가짓수  $= 2^n$

# 데이터 표현 단위와 진수 변환

## ❖ 진법

### ➤ 10진법

- ✓ 0부터 9까지 숫자를 사용하며 10을 한 자리의 기본 단위로 하는 진법
- ✓ 실세계에서 사용하는 수의 표현법

### ➤ 2진법

- ✓ 0과 1의 조합으로 숫자를 표시하는 방법
- ✓ 컴퓨터내부에서 기본적으로 사용하는 방법
- ✓ 0은 전기가 흐르지 않는 상태를 1은 전기가 흐르는 상태로 표시할 수 있으므로 컴퓨터에서 사용하기 편리

### ➤ 8진법

- ✓ 0에서 7까지의 수로 표시하는 것이 8진법

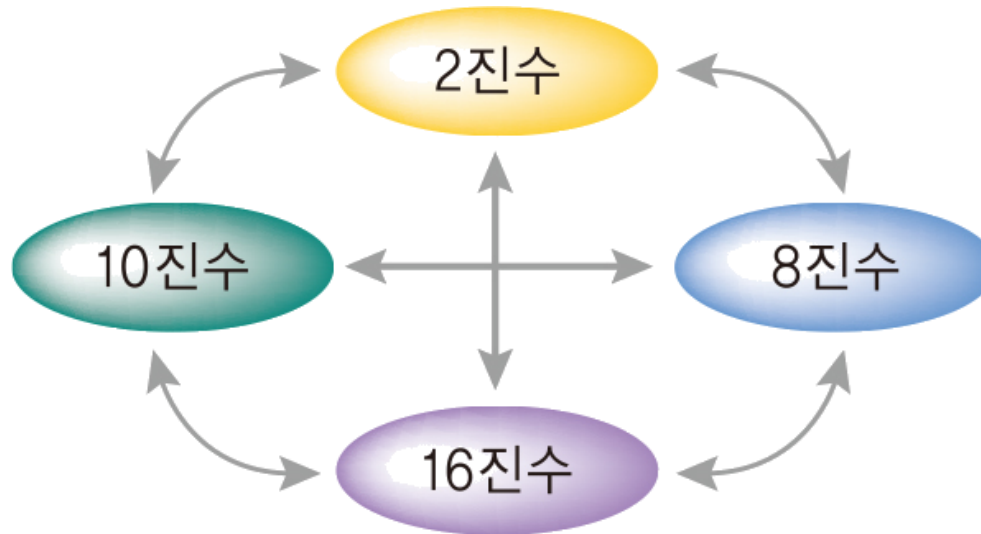
### ➤ 16진법

- ✓ 0부터 9와 A부터 F까지를 사용하여 수를 표시하는 진법  
(10은 A, 11는 B, 12는 C, 13은 D, 14는 E, 15는 F로 표기)
- ✓ 컴퓨터 내부에서는 2진수를 사용하지만 실제로는 2진수를 4자리씩 변환하여 16진수를 사용하는 경우가 많음

# 데이터 표현 단위와 진수 변환

## ❖ 진법 변환

- 주어진 수를 다른 진법으로 변환하는 일반적인 방법은 일단 10진수로 변환한 다음 다른 진법의 수로 바꾸는 것 변환



# 데이터 표현 단위와 진수 변환

## ❖ 진법에 따른 수의 표현

진수	10진수	2진수	8진수	16진수
사용 숫자	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
표현 예	5234 <sub>(10)</sub>	1011 <sub>(2)</sub>	146 <sub>(8)</sub>	5C31 <sub>(16)</sub>

# 데이터 표현 단위와 진수 변환

## ❖ 10진수 변환

### ➤ 2진수를 10진수로 변환하는 방법

2진수

1	0	0	1		0	0	1	1
×	×	×	×		×	×	×	×
$2^7$	$2^6$	$2^5$	$2^4$		$2^3$	$2^2$	$2^1$	$2^0$
128	0	0	16		0	0	2	1

↑ + ↑

10진수 147

### ➤ 2진수를 16진수로 변환한 후 10진수로 변환하는 방법

2진수

1	0	0	1		0	0	1	1
×	×	×	×		×	×	×	×
$2^3$	$2^2$	$2^1$	$2^0$		$2^3$	$2^2$	$2^1$	$2^0$
8	+	0	+	0	+	2	+	1

9 3

16진수

9		3
×		×
$16^1$		$16^0$
144		3

↑ + ↑

10진수 147

10진수

# 데이터 표현 단위와 진수 변환

## ❖ 2진수를 10진수로 변환 예

4비트의 2진수				10진수 변환	10진수
0	0	0	0	$0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	0
0	0	0	1	$0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	1
0	0	1	0	$0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	2
0	0	1	1	$0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	3
0	1	0	0	$0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	4
0	1	0	1	$0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	5
0	1	1	0	$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	6
0	1	1	1	$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	7
1	0	0	0	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	8
1	0	0	1	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	9
1	0	1	0	$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	10 = A
1	0	1	1	$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	11 = B
1	1	0	0	$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$	12 = C
1	1	0	1	$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	13 = D
1	1	1	0	$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	14 = E
1	1	1	1	$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	15 = F

# 데이터 표현 단위와 진수 변환

## ❖ 10진수 변환 예

### ➤ 8진수를 10진수로 변환

✓  $(156)_8 = (110)_{10}$

✓  $1 \times 8^2 + 5 \times 8_1 + 6 \times 8^0 = 110$

### ➤ 2진수를 10진수로 변환

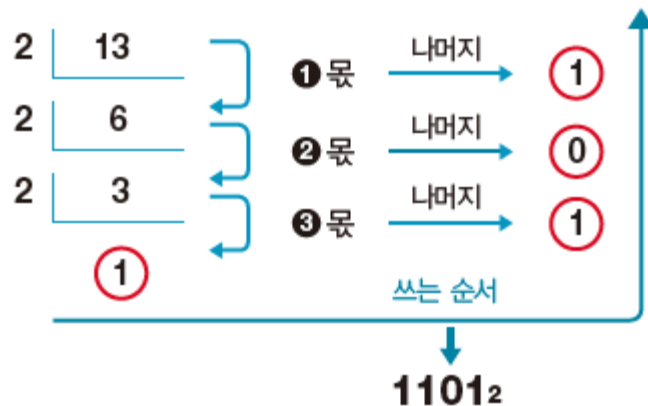
✓  $(11010)_2 = (26)_{10}$

✓  $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26$

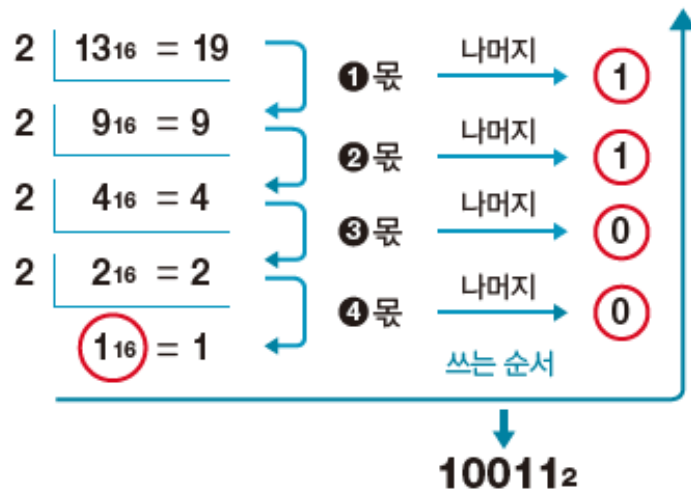
# 데이터 표현 단위와 진수 변환

## ❖ 2진수 변환

➤ 10진수를 2진수로 변환하는 방법



➤ 16진수를 2진수로 변환하는 방법





# 데이터 표현 단위와 진수 변환

## ❖ 10진수를 8진수와 16진수 변환

- 2진수 변환과 유사하게 8진수는 8으로 계속 나누거나 곱하여 변환하고 16진수는 16으로 계속 나누거나 곱하여 변환

8진수로 변경

$$\begin{array}{r} 8 \overline{) 68} \\ 8 \overline{) 8} \quad \text{-----} \quad 4 \\ 1 \quad \text{-----} \quad 0 \end{array}$$

(104)<sub>8</sub>

16진수로 변경

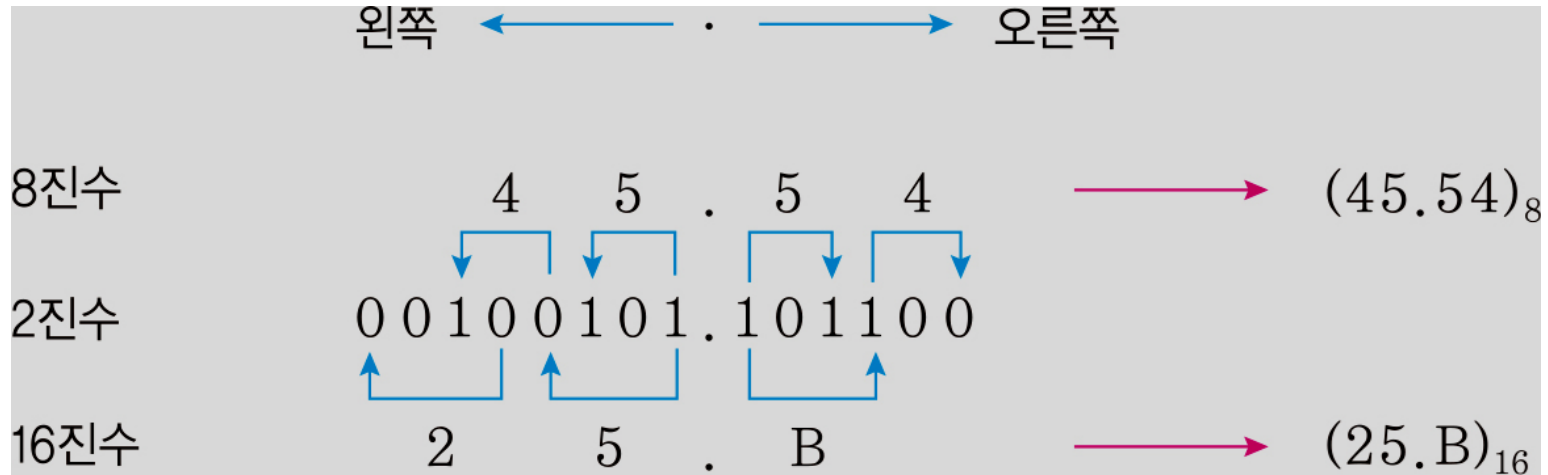
$$\begin{array}{r} 16 \overline{) 68} \\ 4 \quad \text{-----} \quad 4 \end{array}$$

(44)<sub>16</sub>

# 데이터 표현 단위와 진수 변환

## ❖ 2진수, 8진수, 16진수의 상호 변환 관계

- 2진수의 소수점을 기준으로 몇 개씩 묶어나가며 해당 진수로 변환



# 데이터 표현 단위와 진수 변환

## ❖ 16진수와 2진수 변환표

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

# 데이터 표현 단위와 진수 변환

## ❖ 비트, 바이트, 워드

### ➤ 비트(bit)

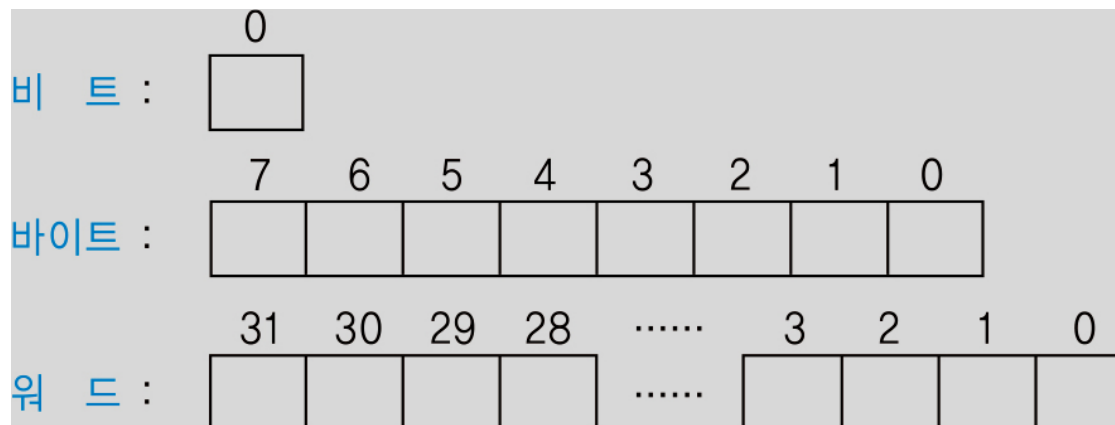
- ✓ 컴퓨터에서 사용하는 최소의 단위로서 0이나 1을 나타냄
- ✓ N비트로 표현할 수 있는 정보는  $2^N$ 개

### ➤ 바이트(byte)

- ✓ IBM S/360에서 8비트 단위로 정리를 처리하면서 8비트를 1바이트로 구성
- ✓ 문자를 나타내는 최소 단위로 영문자나 숫자

### ➤ 워드(word)

- ✓ 명령어나 연산을 처리하는 기본 단위
- ✓ 기억장치에서 한번에 얻을 수 있는 데이터의 양
- ✓ 워드의 크기는 컴퓨터의 종류에 따라 2바이트, 4바이트, 8바이트 등



# 데이터 표현 단위와 진수 변환

## ❖ 바이트, 메가, 기가, 테라, 페타 바이트

- 1KB(Kilo Byte): 1,024바이트( $2^{10}$ 바이트)
- 1MB(Mega Byte): 1,048,576바이트 또는 1,024KB( $2^{20}$ 바이트), 약 백만 바이트
- 1GB(Giga Byte): 1,073,741,824바이트 또는 1,024MB( $2^{30}$ 바이트), 약 십억 바이트
- 1TB(Tera Byte): 1,024GB( $2^{40}$ 바이트), 약 1조 바이트
- 1PB(Peta Byte): 1,024TB( $2^{50}$ 바이트), 약 1,000조 바이트

이름	약어	십진법	이진법
킬로(kilo)	K	$10^3 = 1000^1$	$1024^1 = 2^{10} = 1,024$
메가(mega)	M	$10^6 = 1000^2$	$1024^2 = 2^{20} = 1,048,576$
기가(giga)	G	$10^9 = 1000^3$	$1024^3 = 2^{30} = 1,073,741,824$
테라(tera)	T	$10^{12} = 1000^4$	$1024^4 = 2^{40} = 1,099,511,627,776$
페타(peta)	P	$10^{15} = 1000^5$	$1024^5 = 2^{50} = 1,125,899,906,842,624$
엑사(exa)	E	$10^{18} = 1000^6$	$1024^6 = 2^{60} = 1,152,921,504,606,846,975$
제타(zetta)	Z	$10^{21} = 1000^7$	$1024^7 = 2^{70} = 1,180,591,620,717,411,303,424$

# 데이터 표현 단위와 진수 변환

## ❖ 진수 자동 변환 함수

```
bin(11); bin(0o11); bin(0x11)
oct(11); oct(0b11); oct(0x11)
hex(11); hex(0b11); hex(0o11)
```

### 출력 결과

```
'0b1011'  '0b1001'  '0b10001'
'0o13'    '0o3'     '0o21'
'0xb'     '0x3'     '0x9'
```

# 데이터 표현 단위와 진수 변환

## ❖ 진수 변환

- 숫자를 세는 방법인 2진수, 8진수, 10진수, 16진수 등을 선택하고 값을 입력해 해당 진수별 숫자를 출력

```
1 sel = int(input("입력 진수 결정(16/10/8/2) : "))
2 num = input("값 입력 : ")
3
4 if sel == 16 :
5     num10 = int(num, 16)
6 if sel == 10 :
7     num10 = int(num, 10)
8 if sel == 8 :
9     num10 = int(num, 8)
10 if sel == 2 :
11     num10 = int(num, 2)
12
13 print("16진수 ==> ", hex(num10))
14 print("10진수 ==> ", num10)
15 print(" 8진수 ==> ", oct(num10))
16 print(" 2진수 ==> ", bin(num10))
```

```
Python Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/CookPython/Code03-04.py =====
입력 진수 결정(16/10/8/2) : 16
값 입력 : FF
16진수 ==> 0xff
10진수 ==> 255
 8진수 ==> 0o377
 2진수 ==> 0b11111111
>>>
```

사용자가 입력한 값

Ln: 579 Col: 77

# 기본 데이터형

## ❖ 데이터형

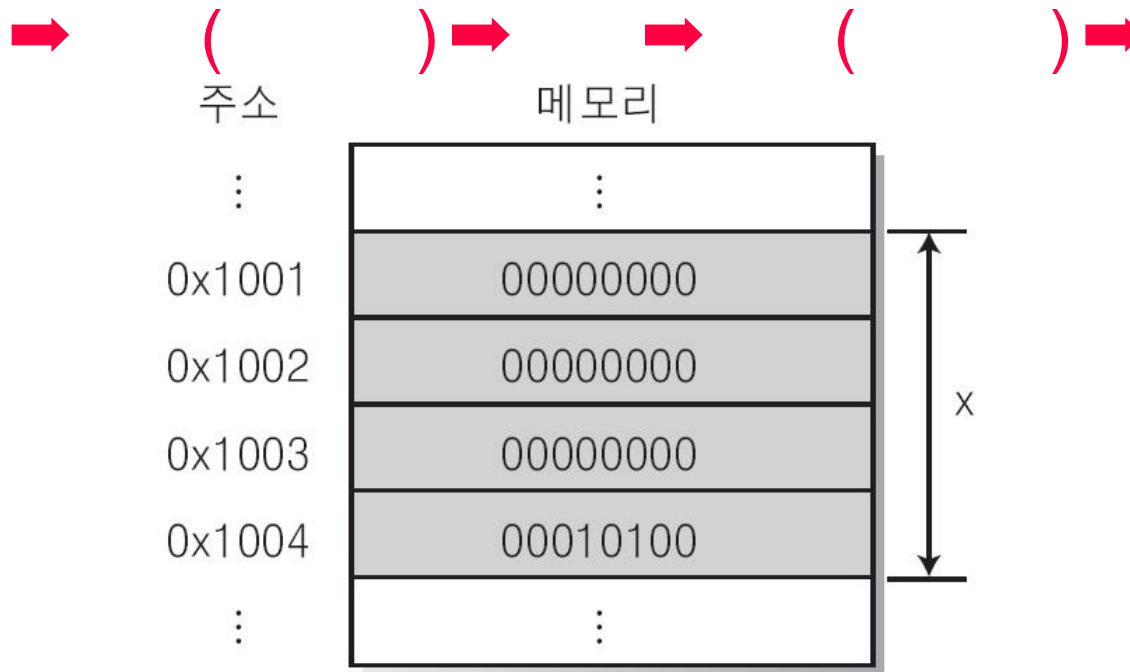
- 프로그래밍 언어에서는 처리할 데이터 또는 수행된 결과를 저장하기 위한 형식을 제공
- 기본형 : int(정수형), float(실수형), complex(복소수형), bool(논리값형)
- 군집형 : str(문자열), list(리스트), tuple(튜플), set(집합), dict(사전)
- 기타 자료형 : bytearray, bytes, frozenset



# 기본 데이터형

## ❖ 데이터형의 의미

- 타입 : 변수가 가질 수 있는 값의 범위와 이 값에 대한 연산들의 집합을 의미
- 영역 : 변수 x의 사용이 허락되는 범위
- 수명 : 변수 x가 메모리 주소에 할당되어 있는 기간



# 기본 데이터형

## ❖ 크기

- 저장하기 위한 메모리의 크기가 고정되어 있기 때문에 값의 범위가 유한
  - ✓ 숫자 타입에서 문제가 발생할 수 있음
- 연산은 유효한 범위를 벗어나는 값을 계산할 수 없음
- 수학의 실수를 근사하기 위하여 고정된 크기의 부동소수점 실수를 사용할 때 발생하는 문제는 더욱 심각
  - ✓ 컴퓨터의 부동소수점 계산은 수학에서의 계산과 일치하지 않을 수 있음

# 기본 데이터형

## ❖ 숫자형(정수형과 실수형)(1/3)

```
a = 123  
type(a)
```

변수에 값을 넣는 순간에 변수의 데이터형이 결정된다

출력 결과

```
<class 'int'>
```

```
a = 100 ** 100  
print(a)
```

int의 크기에는 제한이 없다.

출력 결과

```
10000000~000000
```

# 기본 데이터형

## ❖ 숫자형(정수형과 실수형)(2/3)

```
a = 0xFF  
b = 0o77  
c = 0b1111  
print(a, b, c)
```

정수형에는 16진수, 8진수, 2진수도 사용할 수 있다.

출력 결과

255 63 15

```
a = 3.14  
b = 3.14e5  
print(a, b)
```

실수형은 3.14, -2.7처럼 소수점이 있는 데이터이다,  
또 3.14e5처럼 표현할 수도 있다. 3.14e5는  
3.14\*10<sup>5</sup>을 의미한다.

출력 결과

3.14 314000.0

# 기본 데이터형

## ❖ 숫자형(정수형과 실수형)(3/3)

```
a = 10; b = 20
```

```
print(a + b, a - b, a * b, a / b)
```

 정수 및 실수 데이터형은 사칙 연산 +, -, \*, /를 수행할 수 있다.

출력 결과

```
30 -10 200 0.5
```

```
a, b = 9, 2
```

```
print(a ** b, a % b, a // b)
```

제곱을 의미하는 \*\*, 나머지를 구하는 %,  
나눈 후에 소수점을 버리는 // 연산자도 사용할 수 있다.

출력 결과

```
81 1 4
```

# 기본 데이터형

## ❖ 불형

```
a = True  
type(a)
```

불(Bool)형은 참(True)이나 거짓(False)만 저장할 수 있다.

출력 결과

```
<class 'bool'>
```

```
a = (100 == 100)  
b = (10 > 100)  
print(a, b)
```

불형은 비교의 결과를 참이나 거짓으로 저장하는 데 사용될 수도 있다.

출력 결과

```
True False
```

# 기본 데이터형

## ❖ 문자열(1/2)

```
a = "파이썬 만세"
```

```
a
```

```
print(a)
```

```
type(a)
```

문자열을 'abc' , "파이썬 만세" , "1" 등 문자집합을 의미한다.  
문자열은 양쪽을 큰따옴표( ")나 작은따옴표( ')로 감싸야 한다.

### 출력 결과

```
'파이썬 만세'
```

```
파이썬 만세
```

```
<class 'str'>
```

```
"작은따옴표는 ' 모양이다."
```

```
'큰따옴표는 " 모양이다.'
```

문자열 중간에 작은따옴표나 큰따옴표를 출력하고 싶다면  
다른 따옴표로 묶어 주면 된다.

### 출력 결과

```
"작은따옴표는 ' 모양이다."
```

```
'큰따옴표는 " 모양이다.'
```

# 기본 데이터형

## ❖ 문자열(2/2)

```
a = "이건 큰따옴표 \" 모양."  
b = '이건 작은따옴표 \' 모양.'  
print(a, b)
```

역슬래시(\) 뒤에 큰따옴표나 작은따옴표를 사용해도  
글자로 인식한다.

### 출력 결과

이건 큰따옴표 " 모양. 이건 작은따옴표 ' 모양.

```
a = '파이썬 \n만세'  
print(a)
```

문자열을 여러 줄로 넣으려면  
중간에 \n을 포함시키면 된다.

### 출력 결과

파이썬  
만세

```
a = """파이썬  
만세"""  
a  
print(a)
```

작은따옴표나 큰따옴표 3개를  
연속해서 묶어도 된다.

### 출력 결과

'파이썬\n만세'  
파이썬  
만세