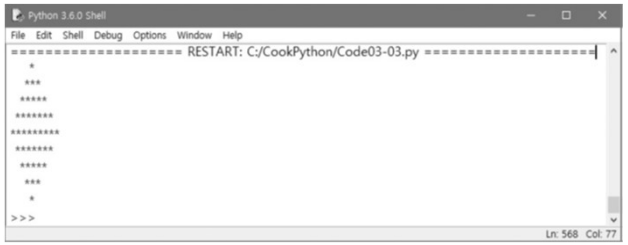
Section01 예제 프로그램

- **[프로그램] 다이아몬드 모양 출력**
  - 다이아몬드 모양의 별표를 출력



Section02 print() 함수를 사용한 다양한 출력

- **print( ) 함수의 서식**

```
print("안녕하세요?")
```
결과는 '안녕하세요?' 이다

```
❶ print("100")
❷ print("%d" % 100)
```
❶의 결과로 나온 100은 숫자 100(백)이 아닌 문자 100(일영영)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 [그림 3-1]과 같이 수정한다.

그림 3-1 서식과 숫자의 대응

Section02 print() 함수를 사용한 다양한 출력

- **print( ) 함수를 사용한 다양한 출력**

```
print("%d / %d = %d" % (100, 200, 0.5))
```
결과는 100/200=0이다.

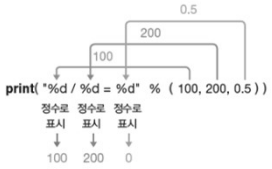


그림 3-2 서식과 숫자의 불일치 상황

표 3-1 print( ) 함수에서 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

따라서 코드를 다음과 같이 수정

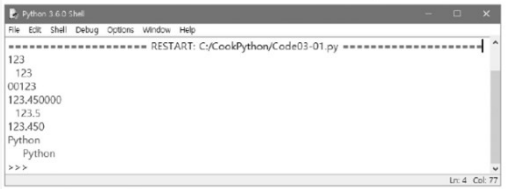
```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```


Section02 print() 함수를 사용한 다양한 출력

- **print( ) 함수를 사용한 깔끔한 출력**

Code03-01.py

```
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")
```



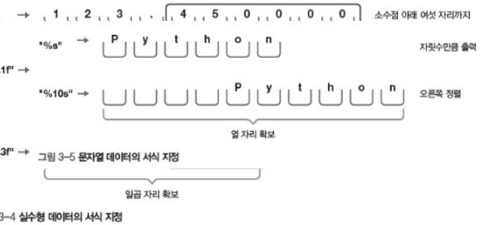


그림 3-4 실수형 데이터의 서식 지정

Section02 print() 함수를 사용한 다양한 출력

- **print( ) 함수를 사용한 깔끔한 출력**
  - format( ) 함수와 { }를 함께 사용해 서식 지정

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

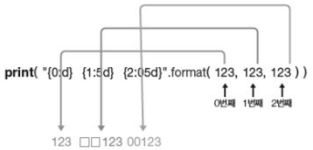


그림 3-6 format( ) 함수의 사용

  - .format을 사용해 출력 순서 지정

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

  - 강제 행 넘기기는 '₩n'을 사용

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


Section02 print() 함수를 사용한 다양한 출력

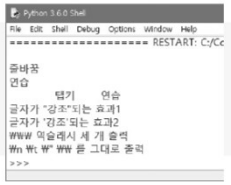
- **print( ) 함수를 사용한 깔끔한 출력**

표 3-2 이스케이프 문자

이스케이프 문자	역할	설명
\n	새로운 줄로 이동	Enter를 누른 효과
\t	다음 탭으로 이동	Tab을 누른 효과
\b	뒤로 한 칸 이동	Backspace를 누른 효과
\\	\ 출력	
\'	' 출력	
\"	" 출력	

Code03-02.py

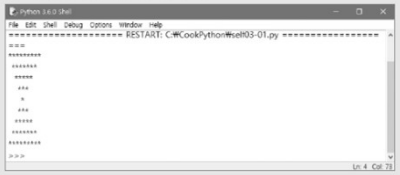
```
1 print("\n줄바꿈\n연습 ")
2 print("\t탭키\t연습")
3 print("글자가 \"강조\"되는 효과1")
4 print("글자가 \'강조\'되는 효과2")
5 print("\\\\\\ 역슬래시 세 개 출력")
6 print(r"\n \t \" \\를 그대로 출력")
```



Section02 print() 함수를 사용한 다양한 출력

SELF STUDY 3-1

별표가 출력되도록 print() 문을 작성해 보자.



Section03 변수의 선언과 사용

- **변수의 선언**
  - 변수는 어떠한 값을 저장하는 메모리 공간(그릇)
  - 변수 선언은 그릇을 준비하는 것
  - 파이썬은 C/C++, 자바 등과는 달리 변수를 선언하지 않아도 되지만 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

그림 3-7 변수의 종류

Section03 변수의 선언과 사용

- Type( ) 함수를 사용하면 변수가 bool(불형), int(정수), float(실수), str(문자열)형으로 생성된 것을 확인할 수 있음

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

출력 결과

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

- **변수명 규칙**
  - 대·소문자를 구분한다(myVar와 MyVar는 다른 변수).
  - 문자, 숫자, 언더바(_)를 포함할 수 있다. 하지만 숫자로 시작하면 안 된다(var2(O), _var(O), var_2(O), 2Var(X)).
  - 예약어는 변수명으로 쓰면 안 된다. 파이썬의 예약어는 True, False, None, and, or, not, break, continue, return, if, else, elif, for, while, except, finally, gloval, import, try 등이다.

Section03 변수의 선언과 사용

- **변수의 사용(1)**
  - 변수는 값을 담으면(대입하면) 사용 가능. 변수에 있던 기존 값은 없어지고 새로 입력한 값으로 변경됨

그림 3-8 변수에 값을 대입해 새로운 값으로 변경된 상태

  - 변수에는 변수의 값을 넣을 수도 있고, 계산 결과를 넣을 수도 있음

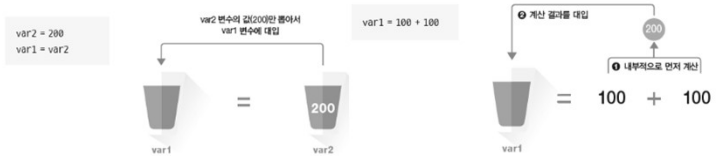


그림 3-9 변수에 변수를 대입하는 방식

그림 3-10 숫자끼리 연산한 결과를 대입하는 방식

Section03 변수의 선언과 사용

- **변수의 사용(2)**
  - 변수에는 숫자와 변수의 연산을 넣을 수도 있음

그림 3-11 변수와 숫자를 연산한 결과를 대입하는 방식

Section03 변수의 선언과 사용

- **변수의 사용(3)**
  - 변수에 연속된 값을 대입하는 방식

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

또는

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

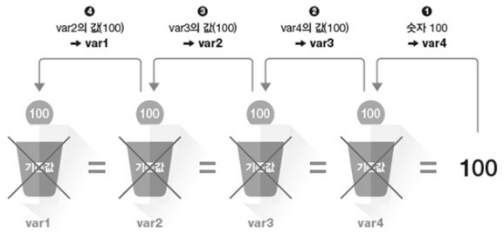


그림 3-12 연속된 값을 대입하는 방식

Section03 변수의 선언과 사용

- **변수의 사용(4)**
  - 변수에 연산 결과를 자신의 값으로 다시 대입하는 방식

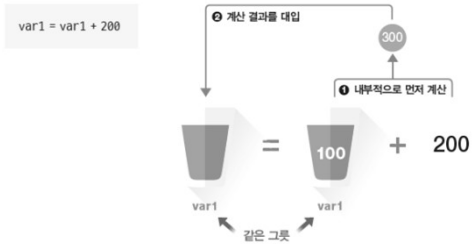


그림 3-13 연산 결과를 자신의 값에 다시 대입하는 방식

Section03 변수의 선언과 사용

- **변수의 사용(5)**
  - 파이썬에서 변수의 데이터 형식은 값을 넣는 순간마다 변경될 수 있는 유연한 구조

```
myVar = 100          # 정수형 변수를 생성(국 그릇 생성)
type(myVar)          # <class 'int'>가 출력
myVar = 100.0        # 이 순간에 실수형 변수로 변경(밥 그릇으로 변경)
type(myVar)          # <class 'float'>가 출력
```

  - 대입 연산자의 왼쪽에는 무조건 변수만 올 수 있고, 오른쪽에는 무엇이든(값, 변수, 수식, 함수 등) 올 수 있음

그림 3-14 왼쪽에 값을 넣을 그릇이 없음

그림 3-15 왼쪽에 값을 넣을 그릇이 있음

Section03 변수의 선언과 사용

여기서 잠깐!

- **동적 타이핑(dynamic typing)**은 변수의 메모리 공간을 확보하는 행위가 실행 시점에서 발생하는 것을 뜻한다. 조금 어렵게 느낄 수도 있지만, 일반적으로 C나 자바는 int data = 8 과 같이 data라는 변수가 정수형이라고 사전에 선언한다. 그에 비해 파이썬은 data = 8 형태로 선언한다. 즉, data라는 변수의 자료형이 정수(integer)인지 실수(float)인지를 프로그래머가 아닌 인터프리터가 스스로 판단하는 것이다. 그리고 그것을 실행 시점에 동적으로 판단하므로 파이썬 언어가 동적으로 자료형의 결정을 지원한다.
- 다른 언어들과 달리 파이썬은 매우 유연한 언어로, 할당받는 메모리 공간도 저장되는 값의 크기에 따라 동적으로 다르게 할당받을 수 있다.

유형	자료형	설명	예	선언 형태
수치형	정수형	양수와 정수	1, 2, 3, 100, −9	data = 1
	실수형	소수점이 포함된 실수	10.2, −9.3, 9.0	data = 9.0
문자형	문자형	따옴표에 들어가 있는 문자형	abc, a20abc	data = 'abc'
논리형	불린형	참 또는 거짓	True, False	data = True

Section04 기본 데이터형

- **숫자형(정수형과 실수형)(1)**

```
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

```
1000000~~~00000
```

Section04 기본 데이터형

- 숫자형(정수형과 실수형)(2)

```
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*105을 의미한다.

출력 결과

```
3.14 314000.0
```


Section04 기본 데이터형

- 숫자형(정수형과 실수형)(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
```


Section04 기본 데이터형

- 불bool형

```
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
```


Section04 기본 데이터형

- 문자열(1)

```
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.'
```

Section04 기본 데이터형

- **문자열(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
```


Section04 기본 데이터형

Code03-05.py

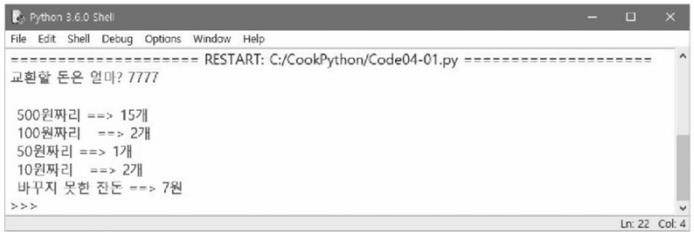
```
1  ## 함수 선언 부분 ##
2  def myFunc() :
3      print('함수를 호출함.')
4
5  ## 전역 변수 선언 부분 ##
6  gVar = 100
7
8  ## 메인 코드 부분 ##
9  if __name__ == '__main__' :
10     print('메인 함수 부분이 실행됩니다.')
11     myFunc()
12     print('전역 변수 값:', gVar)
```

```
def main() :
    print('메인 함수 부분이 실행됩니다.')
    myFunc()
    print('전역 변수 값:', gVar)

if __name__ == '__main__' :
    main()
```

Section05 예제 프로그램

- **[프로그램] 동전교환**



Section 05 산술 연산자

- **산술 연산자의 종류**

표 4-1 산술 연산자의 종류

연산자	의미	사용 예	설명
=	대입 연산자	a = 3	정수 3을 a에 대입
+	더하기	a = 5 + 3	5와 3을 더한 값을 a에 대입
-	빼기	a = 5 - 3	5에서 3을 뺀 값을 a에 대입
*	곱하기	a = 5 * 3	5와 3을 곱한 값을 a에 대입
/	나누기	a = 5 / 3	5를 3으로 나눈 값을 a에 대입
//	나누기(몫)	a = 5 // 3	5를 3으로 나눈 후 소수점을 버리고 값을 a에 대입
%	나머지값	a = 5 % 3	5를 3으로 나눈 후 나머지값을 a에 대입
**	제곱	a = 5 ** 3	5의 3제곱을 a에 대입

Section 05 산술 연산자

- a//b는 a를 b로 나눈 몫이고, a%b는 a를 b로 나눈 나머지값

```
a = 5; b = 3
print(a + b, a - b, a * b, a / b, a // b, a % b, a ** b)
```

출력 결과

```
8 2 15 1.6666666666666667 1 2 125
```

Tip • 세미콜론(;)은 앞뒤를 완전히 분리. 그러므로 a=5; b=3은 다음과 동일하다.
또 콤마(,)로 분리해서 값을 대입할 수도 있어 a, b=5, 3 도 동일

```
a = 5
b = 3
```


Section 05 산술 연산자

▪ 산술 연산자의 우선순위

```
a, b, c = 2, 3, 4
print(a + b - c, a + b * c, a * b / c)
```

출력 결과

```
1 14 1.5
```

```
❶ (a + b) - c
❷ a + (b - c)
```

- 뺄셈에서는 계산되는 순서(연산자 우선순위)가 동일하므로 어떤 것을 먼저 계산하든 동일
- 특별히 괄호가 없을 때는 왼쪽에서 오른쪽 방향으로 계산

Section 05 산술 연산자

▪ 산술 연산자의 우선순위

```
❶ (a + b) * c  ➜  (2 + 3) * 4  ➜  5 * 4  ➜  20
❷ a + (b * c)  ➜  2 + (3 * 4)  ➜  2 + 12  ➜  14
```

- 덧셈(또는 뺄셈)과 곱셈(또는 나눗 셈)이 같이 있으면 곱셈(또는 나눗셈)이 먼저 계산된 후 덧셈(또는 뺄셈)이 계산
- 괄호가 없어도 ❷처럼 계산
- 산술 연산자는 괄호가 가장 우선, 곱셈(또는 나눗셈)이 그 다음, 덧셈(또는 뺄셈)이 마지막
- 덧셈(또는 뺄셈)끼리 있거나 곱셈(또는 나눗셈)끼리 있으면 왼쪽에서 오른쪽으로

Tip • 덧셈, 뺄셈, 곱셈, 나눗셈이 함께 나오면 연산자 우선순위 때문에 종종 혼란스럽게 느껴진다. 이때는 괄호를 사용하면 된다. 괄호를 사용하면 무조건 괄호가 우선 계산, 두 번째 것이 더 나은 코딩

```
❶ a = b + c * d;
❷ a = b + (c * d);
```


Section 05 산술 연산자

▪ 산술 연산을 하는 문자열과 숫자의 상호 변환

- 문자열이 숫자로 구성되어 있을 때, int() 또는 float() 함수 사용해서 정수나 실수로 변환
- 문자열을 int() 함수가 정수로, float() 함수가 실수로 변경

```
s1, s2, s3 = "100", "100.123", "999999999999999999999999"
print(int(s1) + 1, float(s2) + 1, int(s3) + 1)
```

출력 결과

```
101 101.123 1000000000000000000000000
```

Section 05 산술 연산자

- 숫자를 문자열로 변환하려면 str() 함수 사용.
- a와 b가 문자열로 변경되어 100+1이 아닌 문자열의 연결인 '1001'과 '100.1231' 됨

```
a = 100; b = 100.123
str(a) + '1'; str(b) + '1'
```

출력 결과

```
'1001'
'100.1231'
```

Tip • print() 함수는 출력 결과에 작은따옴표가 없어 문자열인지 구분하기가 어려워 사용하지 않음

Section 05 산술 연산자

- **산술 연산자와 대입 연산자**
  - 대입 연산자 = 외에도 +=, -=, *=, /=, %=, //=, **= 사용
  - 예) 첫 번째 대입 연산자 a+=3은 a에 3을 더해서 다시 a에 넣으라는 의미로 a=a+3과 같음

Tip • 파이썬에는 C/C++, 자바 등의 언어에 있는 증가 연산자 ++나 감소 연산자 --가 없음

표 4-2 대입 연산자의 종류

연산자	사용 예	설명
+=	a += 3	a = a + 3과 동일
-=	a -= 3	a = a - 3과 동일
*=	a *= 3	a = a * 3과 동일
/=	a /= 3	a = a / 3과 동일
//=	a //= 3	a = a // 3과 동일
%=	a %= 3	a = a % 3과 동일
**=	a **= 3	a = a ** 3과 동일

Section 05 산술 연산자

- a가 10에서 시작해 프로그램이 진행될수록 값이 누적

```
a = 10
a += 5; print(a)
a -= 5; print(a)
a *= 5; print(a)
a /= 5; print(a)
a //= 5; print(a)
a %= 5; print(a)
a **= 5; print(a)
```

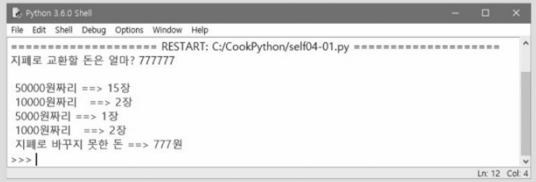
출력 결과

```
15 10 50 10.0 2.0 2.0 32.0
```


Section 05 산술 연산자

SELF STUDY 4-1

돈(예로 777777)을 입력하면 5만 원, 1만 원, 5000원, 1000원 지폐로 교환하는 프로그램을 작성해 보자.



Section 06 관계 연산자

- **관계연산자의 개념**
  - 어떤 것이 크거나 작거나 같은지 비교하는 것(참은 True로, 거짓은 False로 표시)
  - 주로 조건문(if)이나 반복문(while)에서 사용, 단독으로는 거의 사용하지 않음

a 〈 b = { 참 : True / 거짓 : False

그림 4-1 관계 연산자의 기본 개념

표 4-3 관계 연산자의 종류

연산자	의미	설명
==	같다.	두 값이 동일하면 참
!=	같지 않다.	두 값이 다르면 참
〉	크다.	왼쪽이 크면 참
〈	작다.	왼쪽이 작으면 참
〉=	크거나 같다.	왼쪽이 크거나 같으면 참
〈=	작거나 같다.	왼쪽이 작거나 같으면 참

33/38

Section 06 관계 연산자

- a==b를 보면 100이 200과 같다는 의미이므로 결과는 거짓(False)

```
a, b = 100, 200
print(a == b , a != b,  a > b , a < b , a >= b , a <= b)
```

출력 결과

```
False True False True False True
```

- a와 b를 비교하는 관계 연산자 ==를 사용하려다 착오로 =을 하나만 쓴 코드
  - 빨간색 오류로 나타남. a=b는 b 값을 a에 대입하라는 의미이지 관계 연산자가 아님

```
print(a = b)
```

34/38

Section 07 논리 연산자

- **논리 연산자의 종류와 사용**
  - and(그리고), or(또는), not(부정) 세 가지 종류
  - 예) a라는 값이 100과 200 사이에 들어 있어야 한다는 조건 표현

```
(a > 100) and (a < 200)
```

표 4-4 논리 연산자의 종류

연산자	의미	설명	사용 예
and(논리곱)	~이고, 그리고	둘 다 참이어야 참	(a 〉100) and (a 〈200)
or(논리합)	~이거나, 또는	둘 중 하나만 참이어도 참	(a == 100) or (a == 200)
not(논리부정)	~아니다, 부정	참이면 거짓, 거짓이면 참	not(a 〈 100)

35/38

Section 07 논리 연산자

```
a = 99
(a > 100) and (a < 200)
(a > 100) or (a < 200)
not(a == 100)
```

출력 결과

```
False True True
```

- 각 행의 끝에서 Enter 를 2번 눌러야 한다. 첫 번째 1234는 참으로 취급하므로 결과 출력
- 두 번째 0은 거짓이므로 결과가 출력되지 않음. 결론적으로 0은 False, 그 외의 숫자는 모두 True

```
if(1234) : print("참이면 보여요")
if(0) : print("거짓이면 안 보여요")
```

36/38

Section 08 연산자 우선순위

- **연산자 우선순위 : 여러 개의 연산자가 있을 경우 정해진 순서**

표 4-6 연산자 우선순위

우선순위	연산자	의미
1	() [] {}	괄호, 리스트, 딕셔너리, 세트 등
2	**	지수
3	+ - ~	단항 연산자
4	* / % //	산술 연산자
5	+ -	
6	《 》	비트 시프트 연산자
7	&	비트 논리곱
8	^	비트 배타적 논리합
9	|	비트 논리합
10	〈 〉>= <=	관계 연산자
11	== !=	동등 연산자
12	= %= /= //= -= += *= **=	대입 연산자
13	not	논리 연산자
14	and	
15	or	
16	if~else	비교식

37/38