

으뜸 파이썬



2장 변수와 연산자

2.1 파이썬의 출력 함수 print()

- 1장의 내용을 복습해 보자.
- print() 함수를 통해서 파이썬 코드가 수행한 내용을 화면에 출력해 볼 수 있다.
- 대화식 실행 모드와 스크립트 파일 실행 모드가 있다.

- 스크립트를 하나의 파일에 작성 후 일괄적으로 실행

코드 2-1 : 스크립트 코드로 간단한 출력 프로그램 작성하기

print_test.py

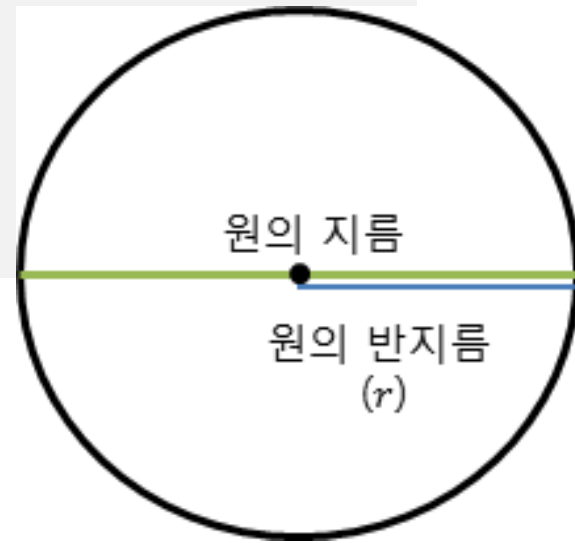
```
print('나의 이름은 :', '홍길동')  
print('나의 나이는 :', 27)  
print('나의 키는', 179, 'cm 입니다.')  
print('10 + 20 =', 10 + 20)  
print('10 * 20 =', 10 * 20)
```

2.2 변수와 친해지기

코드 2-2 : 원의 반지름, 면적, 둘레를 출력하는 프로그램

circle.py

```
print('원의 반지름', 4.0)
print('원의 면적', 3.14 * 4.0 * 4.0)
print('원의 둘레', 2.0 * 3.14 * 4.0 )
```



원의 둘레 = $2\pi r$

원의 면적 = πr^2

- 방금 작성한 프로그램에서 반지름이 5.0, 6.0 인 원의 면적과 둘레를 새로 구하는 경우 다음과 같이 코드를 수정해야 함

```
print('원의 반지름', 5.0)
print('원의 면적', 3.14 * 5.0 * 5.0)
print('원의 둘레', 2.0 * 3.14 * 5.0)
```

```
print('원의 반지름', 6.0)
print('원의 면적', 3.14 * 6.0 * 6.0)
print('원의 둘레', 2.0 * 3.14 * 5.0)
```



번거로운 작업이다
오류의 가능성이 크다

논리 오류 : 문법은 맞지만 의도한 바와 다른 결과
의도한 바는 원의 반지름이 6.0일때의 둘레임

변수 **variable**의 도입

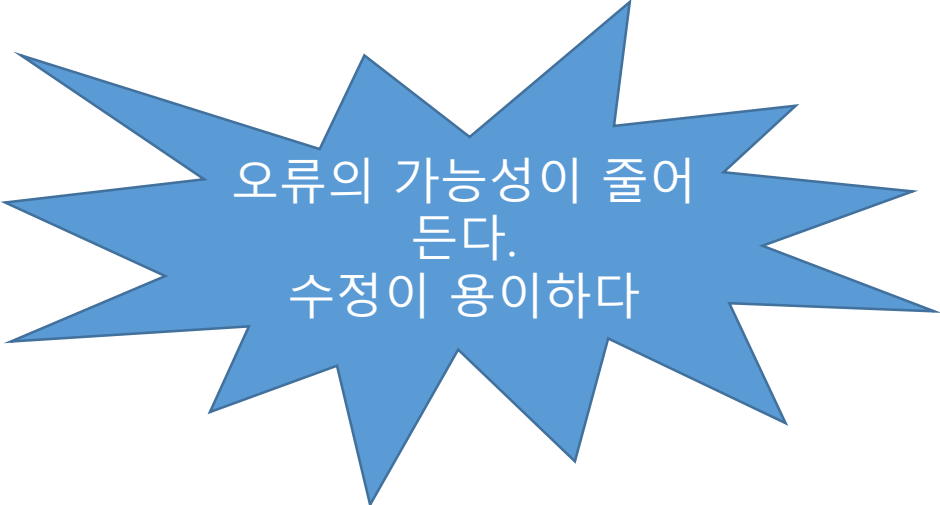
- 프로그램을 작성하다보면 계속해서 값을 변경시켜 주어야하는 경우 하나라도 실수를 하게 되면 프로그램의 오류가 나타남
- **변수**를 도입해서 번거로운 일을 간단하게 만들 수 있음
- 변수에 값을 저장하고 이후에는 값이 아니라 변수를 사용해 보자
 - radius = 4.0 와 같이 radius 라는 이름을 가지는 변수를 만든다
 - 이 변수 radius는 4.0이라는 값을 저장해 두고 있다
 - 나중에 이 변수를 꺼내어 사용할 수 있다

- 변수 `variable` 를 도입

코드 2-3 : 변수를 이용하여 원의 면적과 둘레를 구하는 방법

circle_with_var.py

```
radius = 4.0
print('원의 반지름', radius)
print('원의 면적', 3.14 * radius * radius)
print('원의 둘레', 2.0 * 3.14 * radius)
```



오류의 가능성이 줄어
든다.
수정이 용이하다

- 변수 `variable` 를 도입

코드 2-4 : 변수를 이용하여 원의 면적과 둘레를 구하는 방법

circle_with_var.py (수정)

```
radius = 6.0  
print('원의 반지름', radius)  
print('원의 면적', 3.14 * radius * radius)  
print('원의 둘레', 2.0 * 3.14 * radius)
```

변수에 값을 저장하고 이를 불러서 사용하면
프로그램의 수정이 쉬워지고 오류를 줄일 수 있다

- **변수**variable

- 변할 수 있는 수라는 의미
- 변수(變:변할 변, 數:셀 수)라는 명칭을 사용, '수'는 단순한 수치라기 보다는 **데이터**로 이해하는 것이 더 정확하다.
- 컴퓨터에 값을 저장하는 메모리 위치의 이름
- 이름을 통해 자유롭게 데이터에 대한 읽기, 쓰기, 수정하기가 가능

- **식별자**identifier

- 사용자가 정의하는 변수나 함수에 대해 서로 구별되는 이름을 부여해야 함
- 이와 같이 서로 구별되는 이름을 식별자라고 한다
- 하나의 변수 이름을 여러 개의 메모리 위치를 지칭하는데 사용하게 되면 어느 메모리 공간을 지칭하는지 알기 어려움
- 다른 메모리 위치에는 서로 다른 이름을 부여해야 함

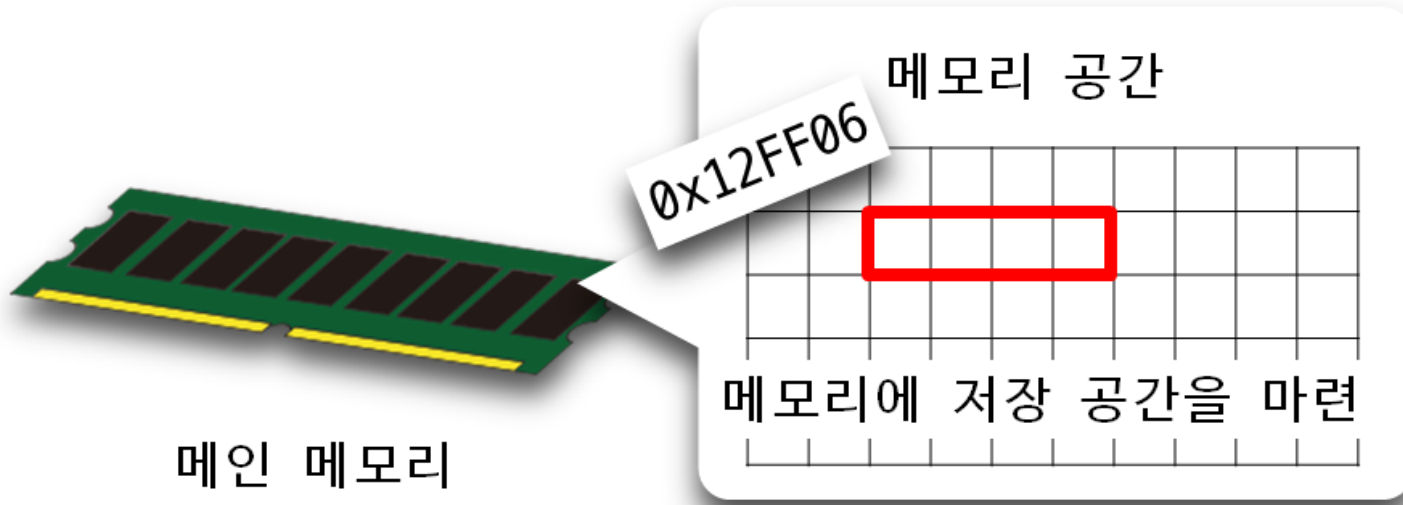
- **메인 메모리** **main memory**

- 컴퓨터의 데이터가 저장되어 읽기와 쓰기, 덮어쓰기를 하는 곳
- 메모리라고도 불림

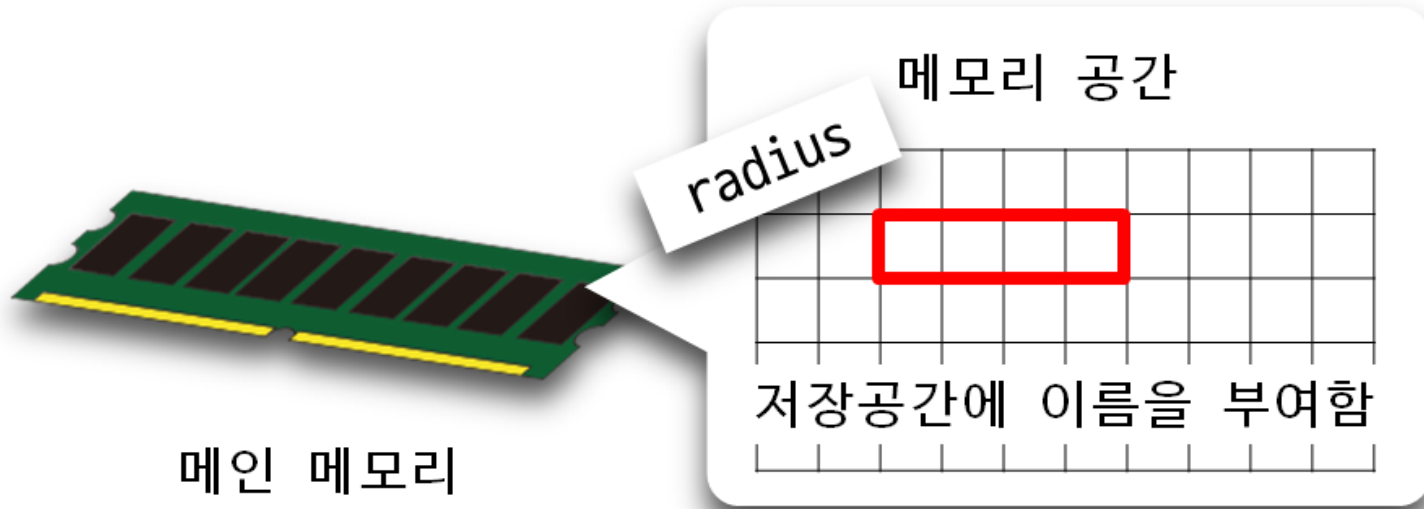
- **메모리 주소** **memory address**

- 메모리에 데이터를 저장한 곳의 위치
- 저장된 데이터를 읽고 쓰기 위해서는 데이터가 저장된 곳(공간 또는 위치)이 어디인가를 알아야 한다
- 주소는 보통 16진수로 표현

컴퓨터의 메인 메모리와 메모리 주소

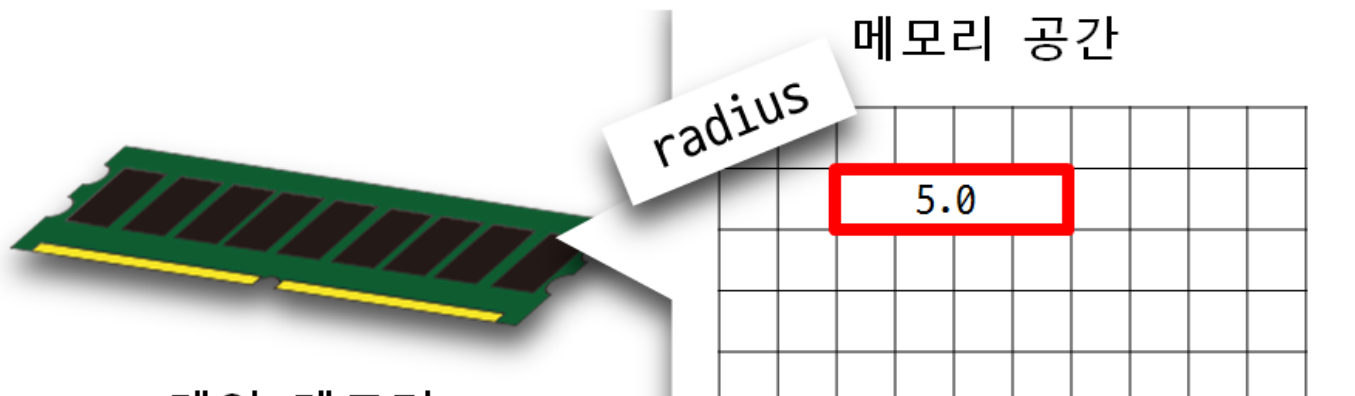


식별을 위한 이름 radius를 부여



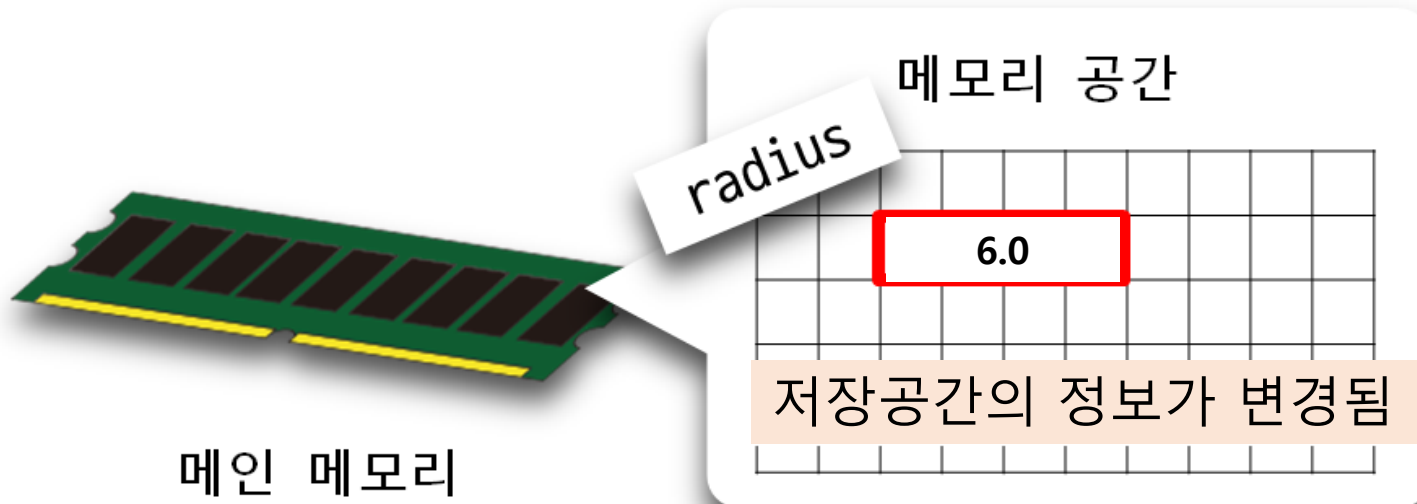
변수와 메모리 공간

radius = 5.0으로 변수에 값을 할당



메인 메모리

radius = 6.0의 결과



메인 메모리

비트와 바이트

- **비트** *bit*

- 컴퓨터에서 사용하는 정보 표현의 최소의 단위
- 0과 1을 이용하여 정보를 표현한다.
- 한 비트만으로 표현 가능한 정보가 너무 적기 때문에 주로 8비트 단위로 저장

- **바이트** *byte*

- 8비트 단위를 바이트라고 함.
- $2^8 = 256$ 가지의 서로 다른 상태 정보를 표현

2.3 변수의 선언

- 리터럴 *literal*
 - 프로그래밍 언어에서 데이터 값을 나타냄

대화창 실습 : 여러 가지 변수의 선언과 출력

```
>>> name = '홍길동'
```

```
>>> print('이름 :', name)
```

이름 : 홍길동

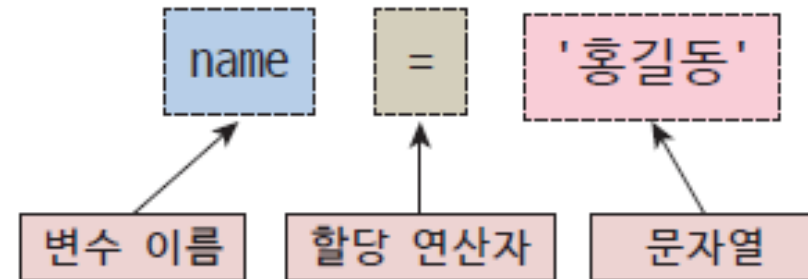
```
>>> width = 10
```

```
>>> height = 5
```

```
>>> rectangle_area = width * height
```

```
>>> print('사각형의 면적 :', rectangle_area)
```

사각형의 면적 : 50



[그림 2-7] 할당 연산자 `=`를 이용하여 변수에 문자열 값을 참조하는 방식



LAB 2-4 : 다음 코드를 입력해 보고 그 출력결과를 밑줄 부분에 적으시오.

```
>>> name = '전우치'
```

```
>>> print('나의 이름은 :', name)
```

```
>>> age = 27
```

```
>>> print('나의 나이는 :', age)
```

```
>>> height = 179
```

```
>>> print('나의 키는', height, 'cm 입니다.')
```

```
>>> sum = 10 + 20
```

```
>>> print('10 + 20 =', sum)
```

```
>>> mult = 10 * 20
```

```
>>> print('10 * 20 =', mult)
```

식별자identifier

- 여러 변수나 함수, 클래스 등을 다른 것들과 구별할 수 있게 지어주는 이름
- 프로그램이 단순할 경우 a, b, n, m,과 같은 단순한 이름의 식별자로도 그 기능을 구현할 수 있다.
- 프로그램이 복잡해지면 walk_distance, num_of_hits, english_dict, student_name과 같이 그 의미를 명확하게 이해할 수 있는 식별자를 사용하는 것이 편리하다.

식별자 이름 규칙

1. 문자와 숫자, 밑줄 문자 _로 이루어진다.
2. 중간에 공백이 들어가면 안 된다.
3. 첫 글자는 반드시 문자나 밑줄 문자 _로 시작해야 한다.
4. 대문자와 소문자는 구분된다. 따라서 Count와 count는 서로 다른 식별자이다.
5. 식별자의 길이에 제한은 없다.
6. 키워드는 식별자로 사용할 수 없다.

[표 2-2] 파이썬에서 사용 가능한 식별자들

사용 가능한 식별자	특징
number4	영문자로 시작하고 난 뒤에는 숫자를 사용할 수 있음
__code__	밑줄 문자는 일반 문자와 같이 식별자 어디든 나타날 수 있음
my_list	
for_loop	키워드라 할지라도 다른 문자와 연결해 쓰면 문제가 없음
높이	유니코드 문자인 한글 문자도 변수로 사용 가능

[표 2-3] 파이썬에서 사용 불가능한 식별자들

사용 불가능한 식별자	사용할 수 없는 이유
1st_variable	숫자 1로 시작하는 식별자임
my list	공백이 들어간 식별자임
global	global은 파이썬의 키워드임
ver2.9	특수 기호가 사용되었음(.)
num&co	특수 기호가 사용되었음(&)

- 키워드 `keyword` 혹은 예약어 `reserved word`
- 이미 예약된 문자로 미리 지정된 역할을 수행하는 단어
- `import`, `for`, `if`, `def`, `class`... 등과 같은 단어가 이에 해당

[표 2-4] 파이썬 키워드 목록: 파이썬 키워드는 사용 용도가 정해져 있어서 변수로 사용할 수 없다.

파이썬의 키워드				
<code>False</code>	<code>class</code>	<code>finally</code>	<code>is</code>	<code>return</code>
<code>None</code>	<code>continue</code>	<code>for</code>	<code>lambda</code>	<code>try</code>
<code>True</code>	<code>def</code>	<code>from</code>	<code>nonlocal</code>	<code>while</code>
<code>and</code>	<code>del</code>	<code>global</code>	<code>not</code>	<code>with</code>
<code>as</code>	<code>elif</code>	<code>if</code>	<code>or</code>	<code>yield</code>
<code>assert</code>	<code>else</code>	<code>import</code>	<code>pass</code>	
<code>break</code>	<code>except</code>	<code>in</code>	<code>raise</code>	

“CapitalizedWords” “mixedCase” 등과 같은 것을 **캡워드** capword 표기법 혹은 **낙타등** camel case 표기법이라고 함

- 좋은 변수 이름을 선택해야 코드를 쉽게 이해할 수 있음

주의 : 변수 이름과 내장 함수 이름

파이썬에서는 sum, max, min, len, list 등은 변수 이름으로 사용할 수 있다. 하지만 이렇게 사용된 변수 이름은 sum(), max(), min(), len(), list() 등과 같은 파이썬의 내장 함수의 이름과 중복되므로 **사용하지 않도록 한다.**

```
>>> sum = 100                # sum()이라는 내장함수 명과 같은 변수이름 sum
>>> lst = [10, 20, 30]
>>> total = sum(lst)         # sum()이라는 내장함수 호출시 오류 발생
...
TypeError: 'int' object is not callable
```

코드 2-5 : 변수에 값을 지정하고 출력하기

variable_test.py

name = '홍길동' ← 문자열 '홍길동'을 저장하는 변수 name

age = 27 ← 정수 값 27을 저장하는 변수 age

print('안녕! 나는', name , '이야. 나는 나이가', age, '살이야.')

코드 2-6 : 변수에 새로운 값을 할당하기

change_var.py

```
name = '홍길동'
```

```
age = 27
```

```
print('안녕! 나는', name , '이야. 나는 나이가', age, '살이야.')
```

```
name = '홍길순'
```

```
age = 23
```

```
print('안녕! 나는', name , '이야. 나는 나이가', age, '살이야.')
```

2.4 변수와 연산자

- 컴퓨터의 자료 값은 덧셈, 뺄셈, 곱셈, 나눗셈들과 같은 **산술 연산** mathematical operation이 가능
- 파이썬은 이러한 산술 연산을 위한 풍부한 연산자를 제공
- "27이라는 값을 age라는 변수에 할당하여라." 라는 명령어와 변수의 할당 과정을 보여주고 있음

```
>>> age = 27
```

1. 27이라는 값을 가지는 정수 객체가 생성된다.

2. age라는 변수가 27이라는 값을 가지는 정수 객체를 참조한다.

객체는 프로그램상의 어떤 자료로 데이터와 함수를 가질 수 있는 것으로 추후 상세히 설명함

- 위의 과정을 통해 정수 27을 age라는 변수명이 참조함
- 할당 연산자의 왼쪽에는 변수 이름이 위치해야 하며, 오른쪽에는 상수 값이나 변수 혹은 수식이 올 수 있음 (반대는 성립하지 않음)
- 할당 연산자가 처음으로 나타나는 경우 “변수 age가 선언^{declare}되었다”라고 함.
- = 기호를 할당 연산자^{assignment operator}라고 한다.
- 아래 문장은 구문오류^{syntax error}가 발생함

```
>>> 27 = age
```

```
File "<stdin>", line 1
```

```
SyntaxError: can't assign to literal
```

- 파이썬은 숫자 값에 대해 같이 기본적으로 사칙연산과 나머지연산, 제곱연산을 수행하는 연산자를 제공

+ : 더하기

- : 빼기

* : 곱하기

/ : 나누기

** : 제곱

% : 나머지

// : 정수 나눗셈 몫

- 파이썬 연산자와 그 의미

[표 2-5] 사칙 연산자와 나머지 연산자, 그리고 거듭 제곱연산자와 동작

연산자	의미	동작
+	덧셈	왼쪽 피연산자와 오른쪽 피연산자를 더한다.
-	뺄셈	왼쪽 피연산자에서 오른쪽 피연산자를 뺀다.
*	곱셈	왼쪽 피연산자와 오른쪽 피연산자를 곱한다.
/	실수 나눗셈	왼쪽 피연산자를 오른쪽 피연산자로 나눈다. 파이썬의 나눗셈은 기본적으로 실수값을 반환한다.
//	정수 나눗셈(몫)	/ 와 달리 나눗셈의 결과를 소수점 이하를 버리고 정수 부분만을 얻고자 할 경우에 사용한다.
%	나머지	모듈로 연산자라고 읽으며 비율을 의미하는 퍼센트와는 상관이 없다. 나눗셈의 나머지를 구한다.
**	거듭제곱	왼쪽 피연산자를 오른쪽 피연산자로 거듭제곱한다.

코드 2-7 : 문자열과 정수의 덧셈연산

number_and_string1.py

```
my_age = 22  
my_height = "177"  
my_age = my_age + 1  
my_height = my_height + 1  
print(my_age, my_height)
```

코드 2-8 : 문자열과 정수의 덧셈연산

number_and_string2.py

```
my_age = 22
my_height = 177
my_age = my_age + 1
my_height = my_height + 1
print(my_age, my_height)
```

- 변수 my_height와 같은 **문자열 형** 변수에 숫자 1을 더하는 연산이 불가능
- 따라서 위의 코드에서는 TypeError라는 오류가 발생
- 연산자는 특정한 자료형에서만 사용이 가능하다

- 정수나 실수 사이에는 덧셈, 뺄셈, 곱셈, 나눗셈의 사칙연산이 잘 적용됨
- 정수에 대해서는 정수 나눗셈과 나머지 연산을 수행 가능
- ** 연산을 사용하여 거듭제곱 연산을 정수와 실수에 대해서도 적용 가능

대화창 실습 : 거듭제곱 연산의 적용

```
>>> 4 ** 0.2
```

```
1.3195079107728942
```

```
>>> 0.2 ** 4
```

```
0.0016000000000000003
```

2.5 자료형의 의미와 자료형 확인

- 자료형 **data type**
- 프로그래밍 언어에서 처리할 수 있는 데이터의 유형
 - 기본 자료형 -부울형, 숫자형(정수, 실수, 복소수), 문자열, 리스트, 튜플, 집합, 딕셔너리
 - 객체가 어떤 자료형인지를 알려주는 `type()`이라는 함수를 제공

대화창 실습 : 다양한 자료형의 이해와 type() 함수

```
>>> num = 85
```

```
>>> type(num)
```

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

```
>>> pi = 3.14159
```

```
>>> type(pi)
```

```
<class 'float'>
```

```
>>> message = "Good morning"
```

```
>>> type(message)
```

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


- 변수 num에는 85라는 정수 값, 변수 pi에는 3.14159라는 실수 값, 변수 message에는 "Good morning"이라는 문자열 값이 각각 할당되어 있음.
- 파이썬의 내장함수 type()을 사용해서 살펴보면 num은 int 클래스, pi는 float 클래스, message는 str 클래스 자료형임을 알 수 있음
- num이라는 변수에 정수 값이 할당되면 변수의 자료형이 int 형으로 결정됨
- 이와 같은 방식으로 자료형이 결정되는 방식을 **동적 형결정** **dynamic typing**이라고 함

동적 형결정과 정적 형결정(용어해설)

- **동적**dynamic - 어떤 행위가 프로그램이 실행되는 도중에 일어나는 것을 의미
- **정적**static - 이와 달리 어떠한 행위가 프로그램이 실행되기 전에 미리 결정되는 것을 의미
- 동적 형결정은 프로그램의 동작이 유연함
- 정적 형결정은 잘못된 값을 넣거나, 서로 연산할 수 없는 데이터를 가지고 연산을 실행하려는 동작을 프로그램 수행전에 소스코드 해석 단계에서 걸러낼 수 있음

파이썬의 자료형을 결정하는 할당 연산자

- `foo = 100`에서 `foo` 변수는 `int` 형을 참조하는 변수
- `foo = 'Hello'`를 통해 `foo`에 문자열을 할당하면 `foo`는 `str` 클래스를 참조하는 변수

대화창 실습 : 동적 형결정(타이핑)의 이해

```
>>> foo = 100
```

```
>>> type(foo)
```

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

```
>>> foo = 'Hello'
```

```
>>> type(foo)
```

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

변수 `foo`는 정수 형 객체를 참조하다가
나중에 문자열 형 객체를 참조할 수 있다
이 변수가 참조하는 자료형은 변경가능하다
이를 동적 형 결정(타이핑)방식이라 한다

정적 타이핑 vs 동적 타이핑

정적 타이핑 static typing	동적 타이핑 dynamic typing
<ul style="list-style-type: none">- 컴파일 시점에 자료형을 검사한다.- 자료의 타입을 일일이 명시해 주어야 한다.- 사용 언어 <p>: C, C++, C#, JAVA, Objective-C, PASCAL 등</p>	<ul style="list-style-type: none">- 자료의 타입을 일일이 알려줄 필요 없어 코드가 간결해 진다.- 반면, 런타임 중 자료형 에러가 날 수 있다.- 사용 언어 <p>: Python, Basic, Ruby, PHP, JavaScript등</p>

대화창 실습 : 다양한 자료형의 이해

```
>>> l = [100, 300, 500, 900]
>>> type(l)
<class 'list'>
>>> d = {'apple': 3000, 'banana': 4200}
>>> type(d)
<class 'dict'>
>>> t = ('홍길동', 30, '율도국의 왕')
>>> type(t)
<class 'tuple'>
```

파이썬은 다양한 자료형을 제공함
리스트, 딕셔너리, 튜플에 대해서는 5, 6장에서
상세하게 알아볼 예정

문자열 변환 함수 str()

- str() 함수는 인수로 입력된 값을 문자열 객체로 만들어서 반환
- 정수형 데이터 값인 숫자 100과, 실수형 데이터 값인 숫자 123.5를 str() 함수의 인자로 넘겨주면 따옴표(")로 둘러싸인 문자열 값이 반환됨
- 리스트형인 ['A', 'B', 'C']를 str()함수의 매개변수로 넘겨줘도 리스트의 요소인 문자들과 혼동되지 않도록 큰따옴표("")로 둘러싸인 문자열 객체가 반환됨

대화창 실습 : str() 함수 실습

```
>>> str(100)
```

```
'100'
```

```
>>> str(123.5)
```

```
'123.5'
```

```
>>> x = ['A', 'B', 'C']
```


```
>>> str(x)
```

```
"['A', 'B', 'C']"
```

```
>>> x = ["A", "B", "C"]
```

```
>>> str(x)
```

```
"['A', 'B', 'C']"
```



str() 함수는 여러가지 자료형의 값을 문자열
형으로 변환시켜 준다

2.6 문자열 자료형

- 연속된 문자로 이루어진 **문자열**`string` 자료형에 대한 처리도 가능
- 문자 하나로 구성된 문자와 여러 문자로 이루어진 문자열을 동일하게 취급
- 작은따옴표('), 큰따옴표(") 모두 사용이 가능

```
>>> txt1 = '강아지 이름은 "햇님"이야'
>>> txt1
'강아지 이름은 "햇님"이야'
```

```
>>> txt2 = "강아지 이름은 '햇님'이야"
>>> txt2
"강아지 이름은 '햇님'이야"
```


2.6 문자열 자료형

- 큰따옴표 내에 “햇님이 좋아!”와 같은 큰따옴표를 가진 문자열을 넣어 주면 에러 발생

```
>>> txt3 = "친구가 "햇님이 좋아!"라고 말했다."
```

```
File "<stdin>", line 1
```

```
    txt3 = "친구가 "햇님이 좋아!"라고 말했다."
```

```
        ^
```

```
SyntaxError: invalid syntax
```



아래와 같이 \" 해야 화면에 따옴표가 출력됨

```
>>> txt3 = "친구가 \"햇님이 좋아!\"라고 말했다."
```

```
>>> txt3
```

```
"친구가 "햇님이 좋아!"라고 말했다."
```

- 문자열은 둘 이상이 연속적으로 나타나거나 중간에 공백 문자나 줄바꿈 문자가 있더라도 이를 하나의 연속적인 문자로 간주

```
>>> txt4 = 'Hello 'Python'  
>>> txt4  
'Hello Python'
```

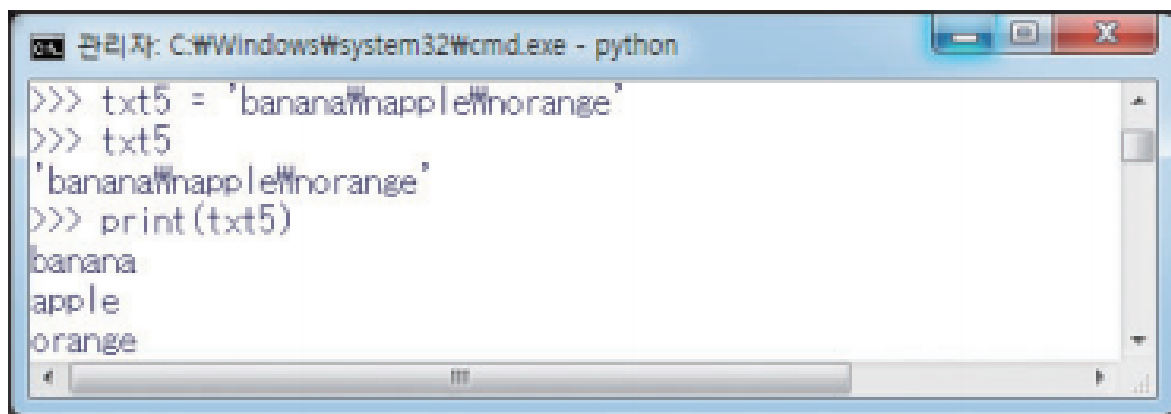
윈도를 비롯한 여러 한글 운영체제에서 역슬래시 문자는 ₩로 나타남. 그러나 IDLE 에서는 아래와 같이 출력됨

- 여러 줄의 문자열을 표현하기 위해서는 **₩n** 문자를 삽입

```
>>> txt5 = 'banana\napple\norange'  
>>> txt5  
'banana\napple\norange'  
>>> print(txt5)  
banana  
apple  
orange
```

- 이스케이프 `escape` 문자
 - `\n`, `\t`
- 파이썬 대화창에서 `\n`이나 `\t`를 가진 문자열을 살펴보면 `"hello\nworld"`나 `"hello\tworld"`와 같이 `\n`, `\t`를 문자 그대로 표현함
- `print()` 함수 내의 입력 값으로 사용시 `\n`은 줄바꿈을 수행
- `\t`는 탭 문자의 삽입 기능을 수행

- 한글 윈도 운영체제의 키보드에서는 이스케이프 문자 역슬래시는 화폐의 단위를 표기하는 원표시(₩)로 나타남



```
>>> txt5 = 'banana₩apple₩orange'
>>> txt5
'banana₩apple₩orange'
>>> print(txt5)
banana
apple
orange
```

[그림 2-9] 한글 윈도에 나타나는 이스케이프 문자의 표시

따옴표 3 개로 둘러싸는 방법

- 줄 바꿈을 포함한 문장을 표현할 때
- 큰따옴표, 작은따옴표가 동시에 포함된 문장을 표현할 때

```
>>> txt6 = '''Let's go'''
```

```
>>> txt6
```

```
"Let's go"
```

```
>>> txt7 = '''큰따옴표(")와 작은따옴표(')를 모두 포함한 문장'''
```

```
>>> txt7
```

```
'큰따옴표(")와 작은따옴표(')를 모두 포함한 문장'
```

```
>>> long_str = """사과는 맛있어
```

```
맛있는 건 바나나
```

```
"""
```

```
>>> long_str
```

```
'사과는 맛있어\n맛있는 건 바나나\n'
```

```
>>> print(long_str)
```

```
사과는 맛있어
```

```
맛있는 건 바나나
```

2.7 수치 자료형

- 정수 `int`
 - 음의 자연수, 0 그리고 자연수를 포함
- 실수 `float`
 - 소수점 이하의 값 포함
- 부울형 `bool`
 - 참과 거짓을 의미하는 True와 False로 이루어짐
- 문자열 `string`형
 - 'Hello', 'World'와 같은 문자열의 집합

대화창 실습 : 자료형과 연산자 실습

```
>>> print(1 + 2)
```

```
3
```

```
>>> print(1.0 + 2.0)
```

```
3.0
```

```
>>> print(1 + 2.0)
```

```
3.0
```

```
>>> print(1 / 2)
```

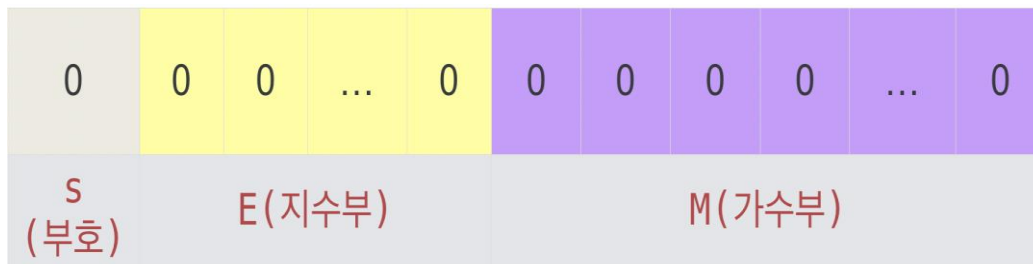
```
0.5
```

```
>>> print(5.0 == 5.00)
```

```
True
```

2.7.1 자료형의 표현 능력과 수치오류

- **부동소수점수** **floating point number**
 - 불가피한 수치 오류를 미세하게 포함



대화창 실습 : 부동소수점 수의 수치오류

```
>>> print(0.1 + 0.1 == 0.2)
```

True

```
>>> print(0.1 + 0.1 + 0.1 == 0.3)
```

False

대화창 실습 : 부동소수점 수의 수치오류 자세히 살펴보기

```
>>> 0.1 + 0.1 + 0.1
```

예상 출력은 0.3

```
0.30000000000000004
```

예상 출력은 0.3이지만 실제로는 아래와 같음

```
>>> 0.1 + 0.1 + 0.1 + 0.1
```

```
0.4
```

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1
```

```
0.5
```

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
```

예상 출력은 0.8

```
0.7999999999999999
```

여기서도 마찬가지로 현상
컴퓨터의 실수 값은 저장될 때 미세한 수치 오류를
포함하고 있음.

```
>>> 0.1 * 8
```

```
0.8
```

```
>>> 0.1 * 19.0
```

```
1.9000000000000001
```

대화창 실습 : 정수 표현의 한계 실습

```
>>> 10 ** 100
```

[illegible]

0000000000000000

파이썬은 아주 큰 정수도 잘 표현한다

대화창 실습 : 정수 표현의 한계

```
>>> 14 / 5
```

```
2.8
```

```
>>> 14 // 5
```

```
2
```

```
>>> 14 % 5
```

```
4
```

```
>>> 14.2 // 5.3          # 14.2를 5.3으로 나눈 몫
```

```
2.0
```

```
>>> 14.2 % 5.3          # 14.2을 5.3으로 나눈 나머지
```

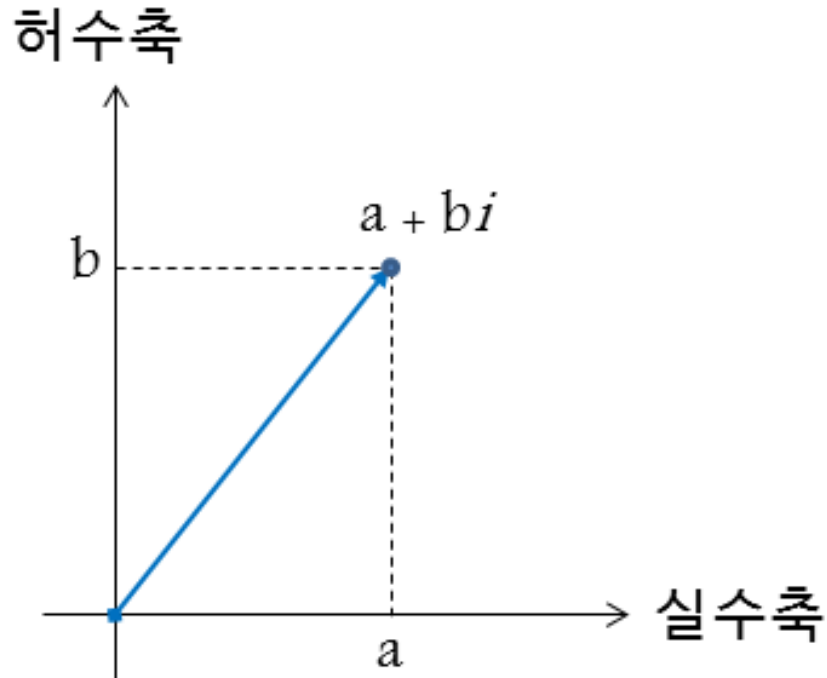
```
3.5999999999999996
```

```
>>> 14.2 - (5.3 * 2.0)   # 14.2 // 5.3 으로 구한 몫이 2.0이므로 나머지는 이런 의미
```

```
3.5999999999999996
```

2.7.2 복소수 자료형

- 실수뿐만이 아닌 허수 값도 가지고 있음
- 기하학적인 좌표로도 표현 가능



- 허수의 표현에 'i'를 사용하면 오류가 발생
- 'j'를 사용해야 복소수 표현이 가능
- real 멤버변수로 실수부를, imag 멤버변수로 허수부를 각각 가져올 수도 있음

대화창 실습 : 복소수 표현 실습

```
>>> c1 = 2 + 3i
```

```
c1 = 2 + 3i
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> c1 = 2 + 3j
```

```
>>> c1.real      # c1의 실수부 출력
```

```
2.0
```

```
>>> c1.imag      # c1의 허수부 출력
```

```
3.0
```

대화창 실습 : 복소수 표현 실습

```
>>> c2 = 5 + 6j
```

```
>>> c3 = c1 + c2
```

```
>>> c3
```

```
(7+9j)
```

켈레 복소수 `complex conjugate`

- 소수의 허수부에 덧셈 역원을 취하여 얻는 복소수
- **켈레 복소수**는 `conjugate()` 메소드로 구할 수 있고, `abs()` 메소드로 해당 복소수의 크기를 알 수 있음

대화창 실습 : 켈레 복소수 표현과 크기 구하기

```
>>> c1 = 2 + 3j
>>> c2 = c1.conjugate()
>>> c2
(2-3j)
>>> abs(c1)
3.605551275463989
```

대화창 실습 : 쉼표 복소수 표현과 크기 구하기

```
>>> c3 = c1 * c2
```

```
>>> c3
```

```
(13+0j)
```

```
>>> abs(c1)*abs(c1)
```

```
12.999999999999998
```


2.8 여러 가지 연산자

연산자	의미	비고
+	덧셈	왼쪽 피연산자와 오른쪽 피연산자를 더한다.
-	뺄셈	왼쪽 피연산자에서 오른쪽 피연산자를 뺀다.
*	곱셈	왼쪽 피연산자와 오른쪽 피연산자를 곱한다
/	실수 나눗셈	왼쪽 피연산자를 오른쪽 피연산자로 나눈다. 파이썬의 나눗셈은 기본적으로 실수값을 반환한다
//	정수 나눗셈의 몫	/ 와 달리 나눗셈의 결과를 정수로 얻고자 할 경우에 사용한다.
%	정수 나눗셈의 나머지	모듈로 연산자라고 읽으며 비율을 의미하는 퍼센트와는 상관이 없다. 나눗셈의 나머지를 구한다.
**	거듭 제곱	왼쪽 피연산자를 오른쪽 피연산자로 거듭제곱한다.

2.8.1 할당 연산자

- 우변의 값을 좌변의 변수에 대입 또는 **할당**^{assign}하라는 의미
- `num1 = num2 = num3 = 200`과 같이 **다중 할당**^{multiple assignment}도 가능

대화창 실습 : 다중 할당과 동시 할당

```
>>> num1 = num2 = num3 = 200      # 다중 할당문
```

```
>>> print(num1, num2, num3)
```

```
200 200 200
```

```
>>> num4, num5 = 300, 400        # 동시 할당문
```

```
>>> print(num4, num5)
```

```
300 400
```

대화창 실습 : 할당 연산 실습

```
>>> result1 = 10 * 20
```

```
>>> result1
```

```
200
```

```
>>> result2 = (2 * 3) - (4 ** 2) / 2
```

```
>>> result2
```

```
-2.0
```

```
>>> 300 = 300
```

등호는 두 값이 같다는 의미가 아님

```
...
```

```
SyntaxError: can't assign to literal
```

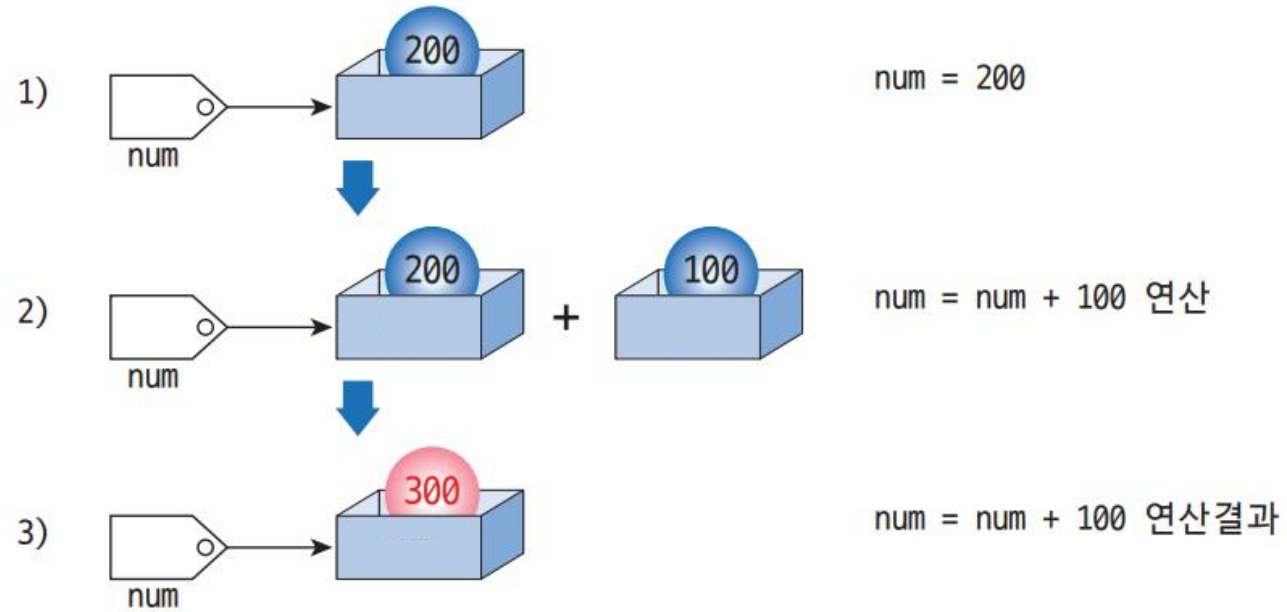
```
>>> str = 'world'
```

```
>>> 'hello' = str
```

리터럴에는 변수를 할당할 수 없다

```
...
```

```
SyntaxError: can't assign to literal
```



[그림 2-12] 덧셈 연산자와 할당 연산자의 수행 과정과 결과

대화창 실습 : 할당 연산 실습

```
>>> num = 200
```

```
>>> num = num + 100
```

200 + 100 연산을 수행하여 그 결과를 num에 할당

```
>>> num
```

300

대화창 실습 : 복합 할당 연산과 그 수행 결과

```
>>> num = 200
```

```
>>> num = num + 100      # 200 + 100 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
300
```

```
>>> num = num - 100      # 300 - 100 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
200
```

```
>>> num = num * 20       # 200 * 20 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
4000
```

```
>>> num = num / 2        # 4000 / 2 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
2000.0
```

[표 2-6] 복합 할당 연산자와 그 의미

연산자	사용 방법	의미
<code>+=</code>	<code>i += 10</code>	<code>i = i + 10</code>
<code>-=</code>	<code>i -= 10</code>	<code>i = i - 10</code>
<code>*=</code>	<code>i *= 10</code>	<code>i = i * 10</code>
<code>/=</code>	<code>i /= 10</code>	<code>i = i / 10</code>
<code>//=</code>	<code>i //= 10</code>	<code>i = i // 10</code>
<code>%=</code>	<code>i %= 10</code>	<code>i = i % 10</code>
<code>**=</code>	<code>i **= 10</code>	<code>i = i ** 10</code>

대화창 실습 : 복합 대입 연산과 그 수행 결과

```
>>> num = 200
```

```
>>> num += 100          # 200 + 100 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
300
```

```
>>> num -= 100          # 300 - 100 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
200
```

```
>>> num *= 20           # 200 * 20 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
4000
```

```
>>> num /= 2            # 4000 / 2 연산을 수행하여 그 결과를 num에 할당
```

```
>>> num
```

```
2000.0
```

2.8.2 비교 연산자

[표 2-7] 파이썬의 비교 연산자와 설명

비교연산자	설명	a = 100, b = 200일 때
==	두 피연산자의 값이 같으면 True를 반환한다.	a == b는 False
!=	두 피연산자의 값이 다르면 True를 반환한다.	a != b는 True
>	왼쪽 피연산자가 오른쪽 피연산자보다 클 때 True를 반환한다.	a > b는 False
<	왼쪽 피연산자가 오른쪽 피연산자보다 작을 때 True를 반환한다.	a < b는 True
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 때 True를 반환한다.	a >= b는 False
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 때 True를 반환한다.	a <= b는 True

대화창 실습 : 비교 연산 (비교의 결과가 True, False로 출력된다)

```
>>> a, b = 100, 200
```

```
>>> a == b
```

```
False
```

```
>>> a != b
```

```
True
```

```
>>> a > b
```

```
False
```

```
>>> a < b
```

```
True
```

```
>>> a >= b
```

```
False
```

대화창 실습 : 비교 연산

```
>>> a > = b
```

```
File "<stdin>", line 1
```

```
a > = b
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> a => b
```

```
File "<stdin>", line 1
```

```
a => b
```

```
^
```

```
SyntaxError: invalid syntax
```

- 주의할 점은 !=에서 !와 =사이에 공백을 넣으면 안되며, >= 연산에서도 >와 = 사이에 공백을 넣으면 안된다.
- =>와 같이 등호와 > 연산자의 순서가 바뀌어도 에러가 뜬다.

2.8.3 논리 연산자 **logical operator**

- and, or, not 이 있음
- 논리 AND, OR, NOT 연산을 통해 True(참)나 False(거짓)중 하나의 값을 가지는 **부울** **bool** 값을 반환

대화창 실습 : 부울형 출력 테스트

```
>>> 10 > 20
```

```
False
```

```
>>> 10 < 20
```

```
True
```

```
>>> bool(9)
```

```
True
```

대화창 실습 : 부울형 출력 테스트

```
>>> bool(-1)
```

```
True
```

```
>>> bool(0)
```

```
False
```

```
>>> bool(None)
```

```
False
```

```
>>> bool('')          # 빈 문자열
```

```
False
```

```
>>> bool('hello')     # 문자열
```

```
True
```

```
>>> bool([])          # 빈 리스트
```

```
False
```

```
>>> bool([10, 20])    # 항목을 가진 리스트
```

```
True
```

- 부울값을 가진 데이터에 대해서 적용할 수 있는 연산이 논리 연산이다.
- 논리 연산은 부울형 자료의 값을 조합하여 새로운 부울값을 만들어내는 것이다.
- 파이썬 논리 연산자가 정확하게 어떤 연산을 하는지 아래 [표 2-8]을 통해 알아보자.

[표 2-8] 파이썬의 and, or, not 논리 연산자와 그 의미

연산자	의미
x and y	x와 y중 거짓(False)이 하나라도 있으면 거짓이 되며 모두 참(True)인 경우에만 참이 된다.
x or y	x나 y중에서 하나라도 참이면 참이 되며, 모두 거짓일 때만 거짓이 된다.
not x	x가 참이면 거짓, x가 거짓이면 참이 된다.

2.9 주석문

- 주석문은 프로그램 내에서 코드의 기능을 설명하는 용도로 사용하는 문장
- 인터프리터가 해석을 하지 않고 건너뛴다

2.9.1. 한 줄 주석 처리하기

- 한 줄 전체를 주석문으로 처리할 경우 문장의 맨 처음에 # 기호 붙임
- 문장 내에서 # 기호가 나타나면 # 기호에서부터 전체 줄의 끝까지가 주석문이 된다.

코드 2-9 : 주석을 넣기 전 코드

circle_with_var.py

```
radius = 4.0
print('원의 반지름', radius)
print('원의 면적 ', 3.14 * radius * radius)
print('원의 둘레 ', 2.0 * 3.14 * radius)
```

주석을 넣기 전/후
아래와 같이 코드에 주석을 넣어 줌으로서 코드를
이해하는 것이 더 쉬워졌다

코드 2-10 : 주석을 넣은 후 코드

circle_with_var_comment.py

```
# 원의 반지름을 radius라는 변수로 정의함
radius = 5.0
# 원의 반지름과 면적, 둘레를 각각 출력
print('원의 반지름', radius)
print('원의 면적 ', 3.14 * radius * radius) # 원의 면적을 구하는 식
print('원의 둘레 ', 2.0 * 3.14 * radius)    # 원의 둘레를 구하는 식
```




NOTE : 주석문의 중요성

주석문은 왜 그렇게 중요한 걸까? 파이썬 명령어는 사람이 이해할 수 있는 말에 가까워 보이지만, 컴퓨터를 위한 명령이다. 자신의 코드를 타인이 수정할 경우 주석문을 보고 코드의 의미를 이해하는 데에 도움이 된다. 또 어려운 코드 옆에 설명을 적어 두면 나중에 그 코드를 다시 볼 때 큰 도움이 된다. 따라서 자신의 기억에 의존하지 않고, 주석문을 통해 문서로 남기는 것은 굉장히 중요하다.

2.9.2 여러 줄 주석 처리하기

- 작은따옴표나 큰따옴표 3개를 연속으로 입력하여 여러 줄을 주석 처리

'''

작은따옴표를 이용하여 여러 줄 주석을 만드는 방법입니다

이 방식으로 주석을 만들면

여러 줄에 걸친 주석을 남길 수 있습니다

'''

"""

혹은 이와 같이 큰따옴표를 이용할 수도 있습니다.

큰따옴표를 사용해도 작은따옴표를 사용하는 것과 동일합니다.

"""

코드 2-11 : 파이썬의 주석문에 대해 알아보는 예제

comment_test1.py

파이썬의 주석문에 대해 알아보는 예제입니다

print("주석은")

print("이 프린트 문을 주석으로 처리하세요")

print("실행되지 않습니다.")

코드 2-13 : 여러줄에 걸친 주석문의 사용

comment_test3.py

```
print("주석은")  
''' print ("이 프린트 문을 주석으로 처리하세요")  
print("실행되지 않습니다.")'''
```

2.10 input() 문과 사용자 입력의 처리

- 사용자로부터 입력을 받는 input() 함수가 제공된다.
- 이 함수는 str형으로 값을 받아들인다.
- 따라서 str형의 입력을 정수로 바꾸고자 할 때는 `age = int(input('나이를 입력하세요:'))` 와 같이 사용

코드 2-15 : input() 함수를 통해 사용자의 입력받기

input_test.py

```
name = input('이름을 입력하세요 : ')      # name은 문자열로 입력받음
print('이름 :', name)
age = int(input('나이를 입력하세요 : '))   # age는 문자열로 입력받아 int 형으로 변환
print('10년 후 나이 :', age + 10)          # 따라서 정수 덧셈 연산이 가능
```

나이는 문자열이 아닌 정수로 저장해 두어야
나중에 + 10과 같은 연산자를 사용할 수 있다.

2. 다음 코드의 수행결과는 무엇인가? _____에 들어갈 알맞은 내용을 적으시오.

```
>>> m = int(input('숫자 m을 입력하세요 : '))
```

```
숫자 m을 입력하세요 : 30
```

```
>>> n = int(input('숫자 n을 입력하세요 : '))
```

```
숫자 n을 입력하세요 : 50
```

```
>>> print('m + n =', m + n)
```

```
_____
```

```
>>> print('m - n =', m - n)
```

```
_____
```

3. 사용자로부터 원의 반지름을 정수로 입력받아 이 원의 면적을 출력하여라(힌트 : 원의 면적은 $(3.14) * (\text{반지름}) * (\text{반지름})$ 로 구할 수 있다).



Questions?