

# AI Agent 전문가 교육과정

미디어 산업의 디지털 전환을 선도할 인공지능 에이전트 개발자 양성



LLM 기초

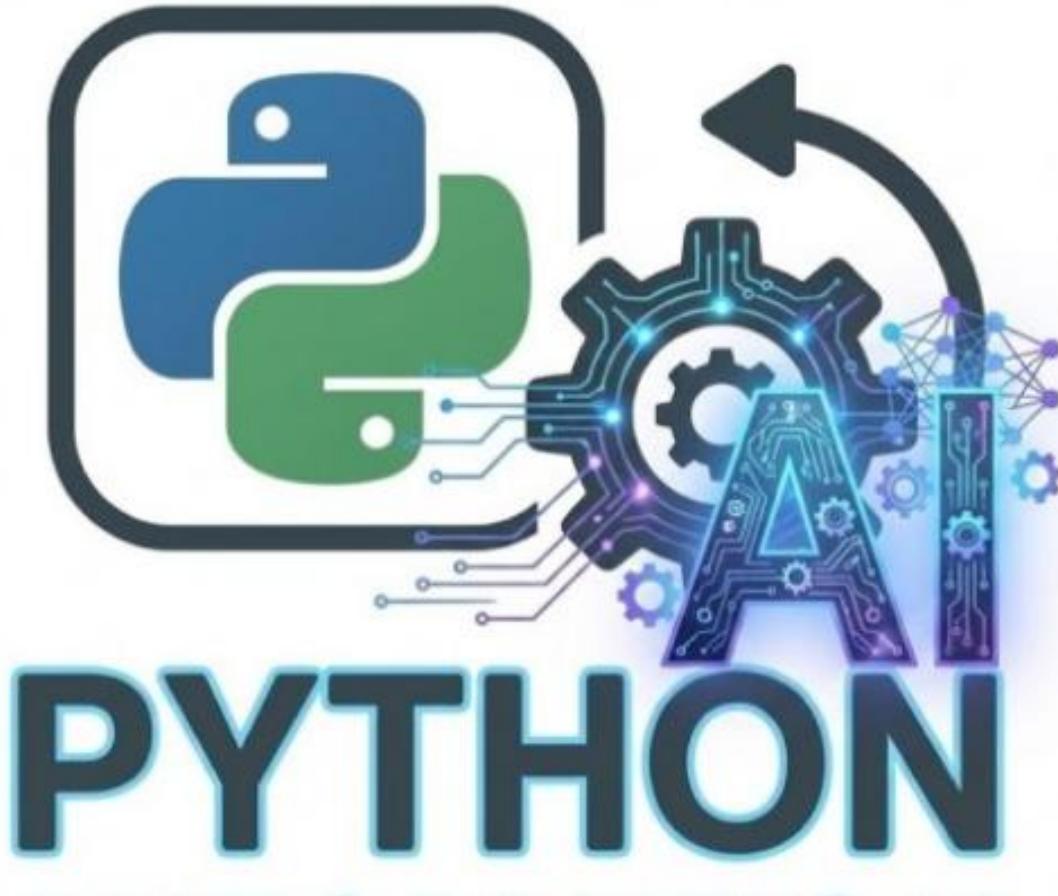


AI Agent 프로그래밍



심화 프로젝트

2025년 12월 일



변수와 연산자

## 학습목표

- `print()` 함수의 사용법을 자세히 알아본다.
- 변수의 개념과 사용법에 대해 이해한다.
- 출력함수를 이용하여 변수의 값을 확인할 수 있으며 다양한 출력을 해 볼 수 있다.
- 값을 그냥 사용하는 것에 비해 변수를 사용하는 것의 장점을 이해한다.
- 변수의 자료형에 대해 이해한다.
- 식별자란 무엇인지 알아보고, 식별자 이름을 짓는 규칙에 대해 알아본다.
- 키워드란 무엇인가 알아본다.
- 연산자의 개념과 종류에 대해 알아본다.
- 다양한 연산자를 활용하여 계산을 수행할 수 있다.
- 문자열을 사용할 수 있다.
- 할당 연산자의 동작을 이해한다.
- 수를 컴퓨터에서 표현하는 방법을 이해한다.
- 비트 단위 연산자를 통해 수를 조작하는 법을 이해한다.
- 주석문의 개념과 사용법에 대해 알아본다.

## 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.

- 대화식 실행모드
  - 파이썬 명령어를 입력할 수 있는 대화식 창에서 즉각적인 반응(피드백)을 받을 수 있는 모드
  - 간단한 코드를 테스트 할 적에는 주로 대화식 실행모드를 사용

```
>>> print('Hello Python!!')  
Hello Python!!
```
- 스크립트 실행모드
  - .py 라는 확장자를 가지는 스크립트를 만들어서 파이썬 인터프리터를 통해 실행
  - 복잡한 로직이 있는 코드는 스크립트 파일을 만들어서 실행

# 다음 코드를 입력해 봅시다

```
>>> print(Hello Python!!)
```

...

```
SyntaxError: invalid syntax
```

오류 발생

**SyntaxError : 구문오류**

**invalid syntax : 유효하지 않은 구문**

번역기는 여러분에게 친절하게 오류를 표시해 주고 알려줍니다

```
>>> print('My age is', 20)
```

```
My age is 20
```

```
>>> print('오늘의 걸음 수', 8000, '걸음')
```

```
오늘의 걸음 수 8000 걸음
```

제대로 된 출력방식

문자열과 숫자는 쉼표로 구분해 줍시다

```
>>> print('Hello ' * 2)
```

```
Hello Hello
```

```
>>> print('Hello ' * 4)
```

```
Hello Hello Hello Hello
```

문자열에 \* 연산을 하고 숫자를 넣어 줄 경우 :  
숫자만큼 문자열을 반복 출력한다

# 2.1 파이썬의 출력 함수

- 대화식 실행모드 제공

	대화식 실행모드	스크립트 실행모드(print_name.py 파일)
입력	>>> print('당신의 이름은 :') 당신의 이름은 : >>> name = '홍길동' >>> print(name) 홍길동	print('당신의 이름은 :') name = '홍길동' print(name)
수행	입력 후 엔터키를 입력하면 수행 됨	\$ python print_name.py 당신의 이름은 : 홍길동

- 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)
```

실행결과

나의 이름은 : 홍길동  
나의 나이는 : 27  
나의 키는 179 cm입니다  
10 + 20 = 30  
10 \* 20 = 200

## 2.2 변수와 친해지기

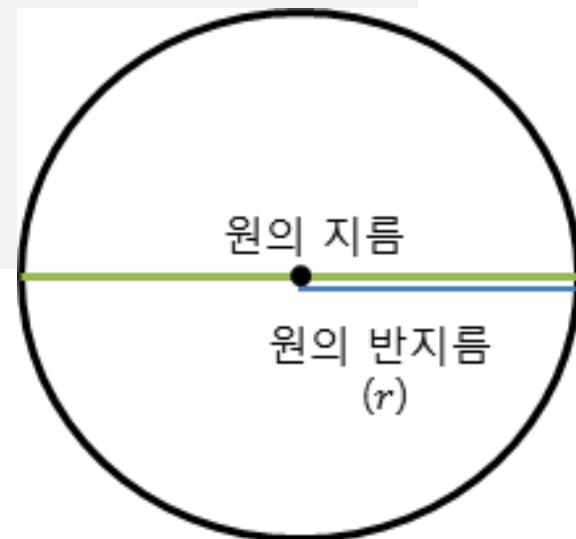
코드 2-2 : 원의 반지름, 면적, 둘레를 출력하는 프로그램

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 )
```

실행결과

```
원의 반지름 4.0  
원의 면적 50.24  
원의 둘레 25.12
```



$$\text{원의 둘레} = 2\pi r$$

$$\text{원의 면적} = \pi r^2$$

- 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)
```

# 변수variable의 도입

- 프로그램을 작성하다보면 계속해서 값을 변경시켜 주어야하는 경우 하나라도 실수를 하게 되면 프로그램의 오류가 나타남
- 변수를 도입해서 번거로운 일을 간단하게 만들 수 있음
- 변수에 값을 저장하고 이후에는 값이 아니라 변수를 사용해 보자
  - radius = 4.0 와 같이 radius라는 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)
```

실행결과

원의 반지름 4.0

원의 면적 50.24

원의 둘레 25.12

- 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

실행결과

```
원의 반지름 6.0  
원의 면적 113.03999999999999  
원의 둘레 37.68
```

- **변수***variable*

- 변할 수 있는 수라는 의미
- 변수(變:변할 변, 數:셀 수)라는 명칭을 사용, '수'는 단순한 수치라기 보다는 데이터로 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

# 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

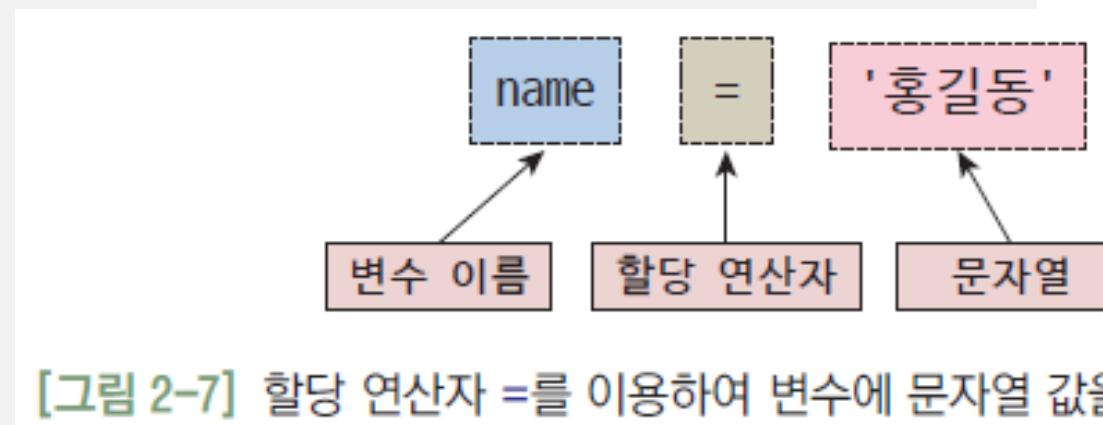
## 2.3 변수의 선언

- 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

# 식별자 identifier

- 여러 변수나 함수, 클래스 등을 다른 것들과 구별할 수 있게 지어주는 이름
- 프로그램이 단순할 경우 a, b, n, m과 같은 단순한 이름의 식별자로도 그 가능을 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. **키워드**는 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				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

"CapitalizedWords" "mixedCase" 등과 같은 것을 **캡워드**<sup>capword</sup> 표기법 혹은 **낙타등**<sup>camel case</sup> 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.')
```

실행결과

안녕! 나는 홍길동 이야. 나는 나이가 27 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.')
```

실행결과

안녕! 나는 홍길동 이야. 나는 나이가 27 살이야.

안녕! 나는 홍길순 이야. 나는 나이가 23 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 operator라고 한다.
- 아래 문장은 구문오류 syntax error가 발생함

```
>>> 27 = age  
File "<stdin>", line 1  
SyntaxError: can't assign to literal
```

- 파이썬은 숫자 값에 대해 같이 기본적으로 **사칙연산**과 **나머지연산**, **제곱연산**을 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.

- 기본 연산자들을 사용해 파이썬을 계산기로 사용  
4 + 10과 같은 문장을 표현식 expression이라고 한다

대화창 실습 : 파이썬 표현식의 사용

```
>>> 4 + 10 # 덧셈 연산
```

```
14
```

```
>>> 4.0 - 0.1 # 뺄셈 연산
```

```
3.9
```

```
>>> 20 * 20 # 곱셈 연산
```

```
400
```

```
>>> 11 / 2 # 실수 나눗셈 연산
```

```
5.5
```

```
>>> 11 // 2 # 정수 나눗셈 연산 - 몫을 구함
```

```
5
```

```
>>> 11 % 2 # 정수 나눗셈 후의 나머지 연산
```

```
1
```

- 기본 연산자들을 사용해 파이썬을 계산기로 사용  
4 + 10과 같은 문장을 표현식 expression이라고 한다

대화창 실습 : 파이썬 표현식의 사용

```
>>> 4 ** 0.5 # 거듭제곱 - 4의 제곱근을 구함
```

```
2.0
```

```
>>> 4 ** 5 # 거듭제곱 - 4의 5승을 구함
```

```
1024
```

## 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)
```

### 실행결과

```
...  
my_height = my_height + 1  
TypeError: must be str, not int  
혹은  
TypeError: can only concatenate str (not "int") to str
```

- 변수 my\_height와 같은 **문자열 형** 변수에 숫자 1을 더하는 연산이 불가능
- 따라서 위의 코드에서는 TypeError라는 오류가 발생
- 연산자는 특정한 자료형에 서만 사용이 가능하다

## 코드 2-8 : 실수와 정수의 덧셈 연산

number\_and\_string2.py

```
my_age = 22  
my_height = 177.5  
my_age = my_age + 1  
my_height = my_height + 1  
print(my_age, my_height)
```

실행결과

23 178.5

- 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  
0.001600000000000003
```

## 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

## 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'>
```

# 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 IDLE에서는 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 \n이나 \t를 가진 문자열을 살펴보면 "hello\nworld"나 "hello\tworld"와 같이 \n, \t를 문자 그대로 표현함
- print() 함수 내의 입력 값으로 사용시 \n은 줄바꿈을 수행
- \t는 탭 문자의 삽입 기능을 수행

# 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  
"""\n  
>>> 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
```

---

## 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
```

3.599999999999996

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

3.599999999999996

## 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 = 200과 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
```

```
...
```

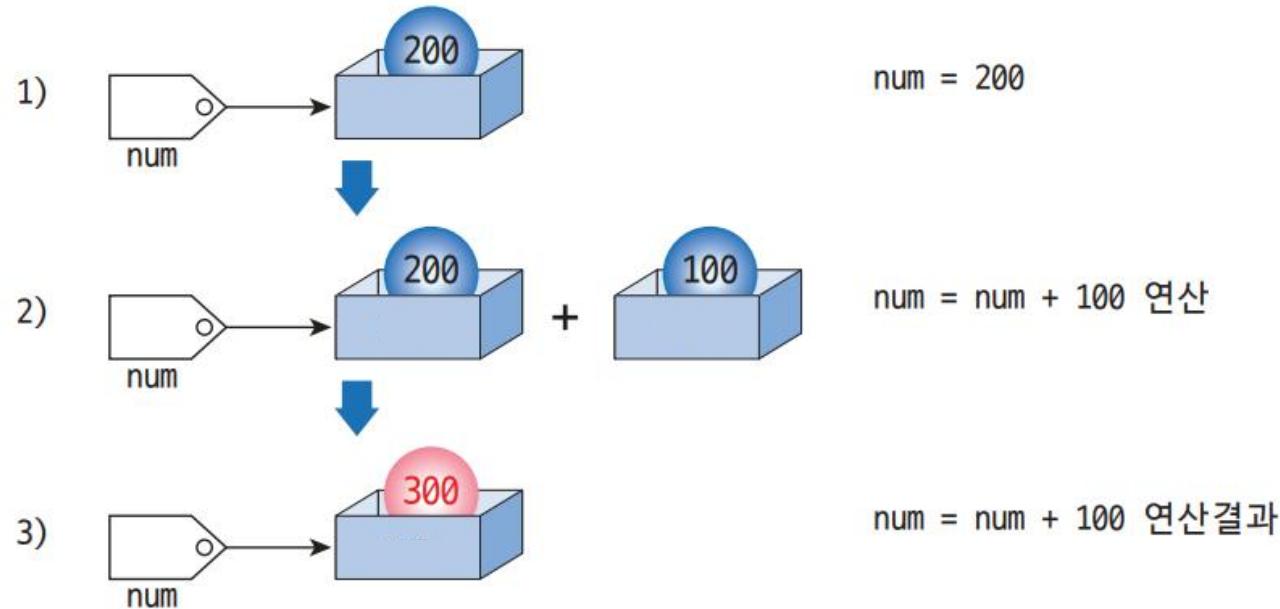
SyntaxError: can't assign to literal

```
>>> 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 : 할당 연산자와 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 할당 연산자와 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	$a = 100, b = 200$ 일 때
$==$	두 피연산자의 값이 같으면 True를 반환한다.	$a == b$ 는 False
$!=$	두 피연산자의 값이 다르면 True를 반환한다.	$a != b$ 는 True
$>$	왼쪽 피연산자가 오른쪽 피연산자보다 클 때 True를 반환한다.	$a > b$ 는 False
$<$	왼쪽 피연산자가 오른쪽 피연산자보다 작을 때 True를 반환한다.	$a < b$ 는 True
$\geq$	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 때 True를 반환한다.	$a \geq b$ 는 False
$\leq$	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 때 True를 반환한다.	$a \leq b$ 는 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

---

## 2.8.3 논리 연산자logical operator

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

대화창 실습 : 부울형 출력 테스트

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

```
False
```

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

```
True
```

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

```
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
<code>x and y</code>	<code>x</code> 와 <code>y</code> 중 거짓(False)이 하나라도 있으면 거짓이 되며 모두 참(True)인 경우에만 참이 된다.
<code>x or y</code>	<code>x</code> 나 <code>y</code> 중에서 하나라도 참이면 참이 되며, 모두 거짓일 때만 거짓이 된다.
<code>not x</code>	<code>x</code> 가 참이면 거짓, <code>x</code> 가 거짓이면 참이 된다.

## 대화창 실습 : 논리 연산 실습

```
>>> x = True
```

```
>>> y = False
```

```
>>> x and y
```

False

```
>>> x or y
```

True

```
>>> not x
```

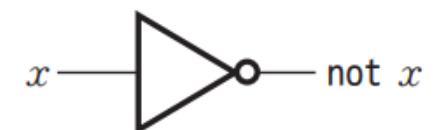
False



$x \text{ and } y$



$x \text{ or } y$



$\text{not } x$

$x$	$y$	$x \text{ and } y$
False	False	False
False	True	False
True	False	False
True	True	True

$x$	$y$	$x \text{ or } y$
False	False	False
False	True	True
True	False	True
True	True	True

$x$	$\text{not } x$
False	True
True	False

[그림 2-13] 논리 회로와 and, or, not 연산

## 2.9 주석문

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

## 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.

## 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.")
```

### 실행결과

```
주석은  
이 프린트 문을 주석으로 처리하세요  
실행되지 않습니다.
```

## 코드 2-12 : 파이썬의 주석문에 대해 알아보는 예제

comment\_test2.py

```
# 파이썬의 주석문에 대해 알아보는 예제입니다  
print("주석은")  
# print("이 프린트 문을 주석으로 처리하세요")  
print("실행되지 않습니다.")
```

실행결과

```
주석은  
실행되지 않습니다.
```

## 코드 2-13 : 여러줄에 걸친 주석문의 사용

comment\_test3.py

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

실행결과

주석은

## 코드 2-14 : 주석문의 잘못된 사용

comment\_test4.py

```
# 아래의 예제는 주석문의 잘못된 사용 예입니다.  
print("주석은")  
''' print("이 프린트 문을 주석으로 처리하세요") '''  
print("실행되지 않습니다.")
```

실행결과

SyntaxError: EOF while scanning triple-quoted string literal

## 2.10 input() 문과 사용자 입력의 처리

- 사용자로부터 입력을 받는 `input()` 함수가 제공된다.
- 이 함수는 `str`형으로 값을 받아들인다.
- 따라서 `str`형의 입력을 정수로 바꾸고자 할때는 `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
```

실행결과

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

이름을 입력하세요 : 홍길동

이름 : 홍길동

나이를 입력하세요 : 22

10년 후 나이 : 32