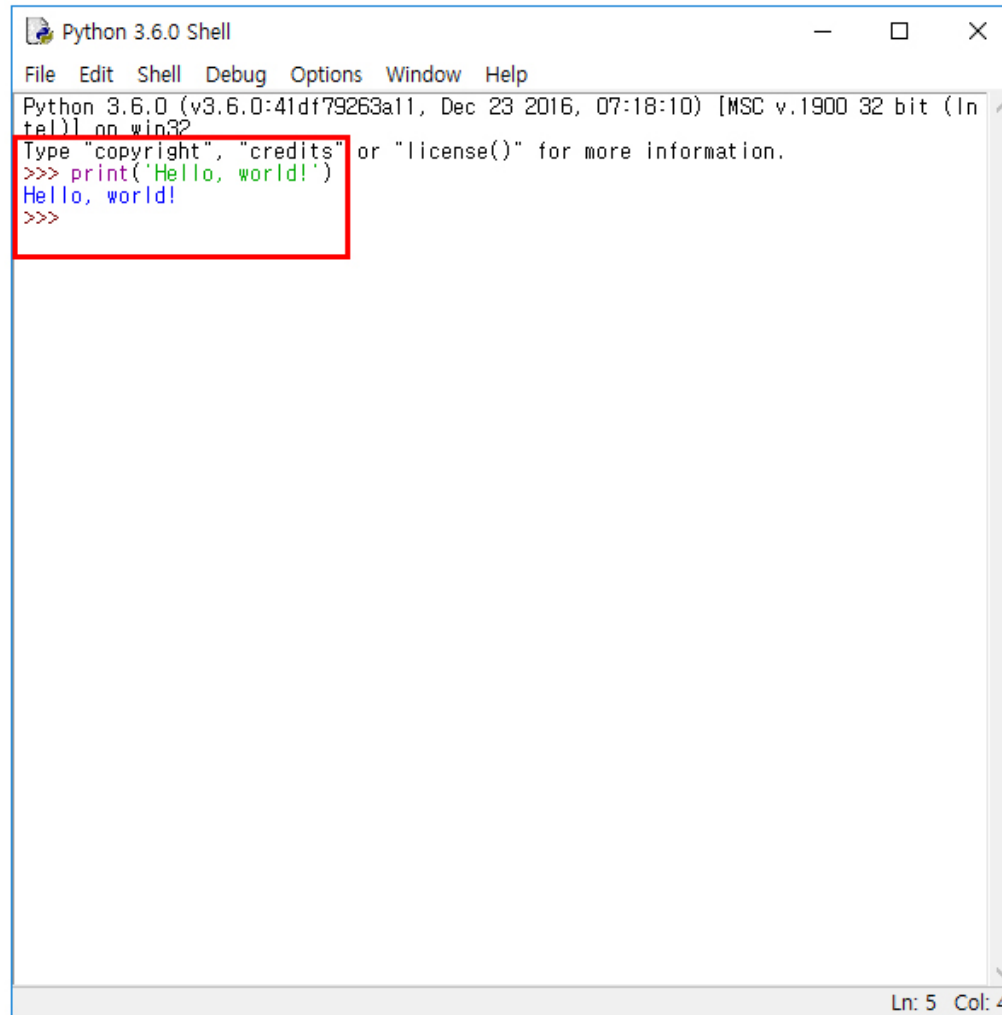


Ch2. 기본 문법

- **Hello, world!로 시작하기**
 - 파이썬을 설치했으니 이제부터 본격적으로 파이썬을 배워보기
 - 프로그래밍 언어의 첫 관문인 Hello, world! 출력부터 알아보기
- IDLE에서 Hello, world! 출력해보기
 - IDLE의 >>> 부분에 다음 내용을 입력한 뒤 엔터 키를 누르기

```
>>> print('Hello, world!')  
Hello, world!  
>>>
```

IDLE의 파이썬 셸에서 Hello, world! 출력



The screenshot shows a window titled "Python 3.6.0 Shell" with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The main text area displays the following content:

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('Hello, world!')
Hello, world!
>>>
```

A red rectangular box highlights the input and output of the `print` statement. The status bar at the bottom right indicates "Ln: 5 Col: 4".

- **IDLE에서 Hello, Python 출력해보기**

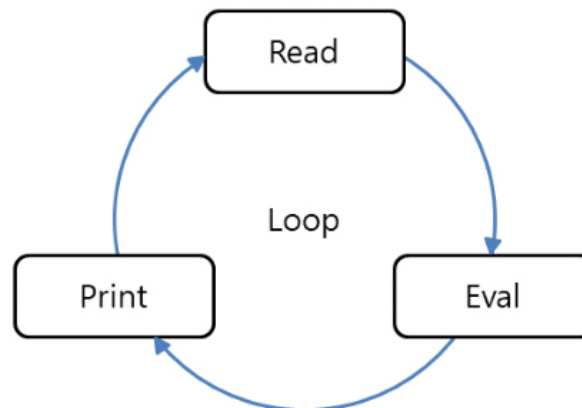
- print() 안에 'Hello, Python'을 넣으면 됨

```
>>> print('Hello, Python')
Hello, Python
>>>
```

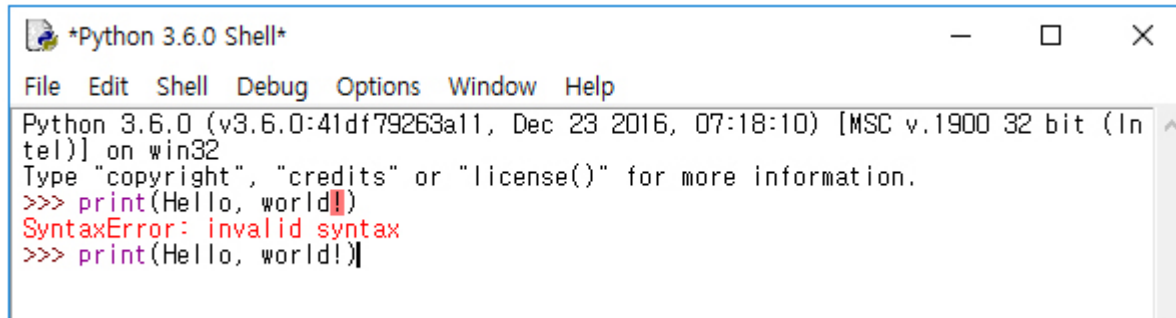
- **에러**

- NameError: name ... is not defined: 함수 이름을 잘못 입력했을 때 발생하는 에러
- 파이썬은 대소문자를 구분하므로 대소문자를 정확히 입력
- print처럼 전부 소문자로 입력했는지 확인
- SyntaxError: invalid syntax: print()안에 Hello, world!를 넣을 때 ' '(작은따옴표)로 묶지 않아서 발생하는 구문 에러(syntax error)
- 작은따옴표로 묶어줌
- SyntaxError: EOL while scanning string literal: 따옴표를 잘못 사용했을 때 발생하는 구문 에러
- 'Hello, world!'와 같이 앞 뒤로 작은따옴표 쌍이 맞는지 확인

- **IDLE에서 Hello, world! 출력해보기**
 - 코드를 한 줄 한 줄 실행하여 결과를 얻는 방식을 인터프리터(interpreter) 방식
 - IDLE처럼 파이썬 코드를 직접 입력해서 실행하는 프로그램을 파이썬 셸(Python Shell)이라고 함
 - >>> 부분을 파이썬 프롬프트(Python prompt)라고 부름
- **대화형 셸**
 - 파이썬 셸은 파이썬 인터프리터와 대화하듯이 코드를 처리한다고 해서 대화형 셸(interactive shell) 또는 인터랙티브 모드(interactive mode)라고도 함
 - 이런 방식을 코드를 읽고, 평가(계산, 실행)하고, 출력한다고 해서 REPL(Read-Eval-Print Loop)이라고 함



- **IDLE의 파이썬 셸에서 에러가 났을 때**
 - IDLE의 파이썬 셸에서 코드를 잘못 입력하여 에러가 나면 올바른 코드를 다시 입력
 - IDLE의 파이썬 셸에서는 ↑ 방향 키를 누르면 이전 코드로 쉽게 돌아갈 수 있음
 - 이전 코드에서 엔터 키를 누르면 해당 코드를 다시 사용할 수 있음

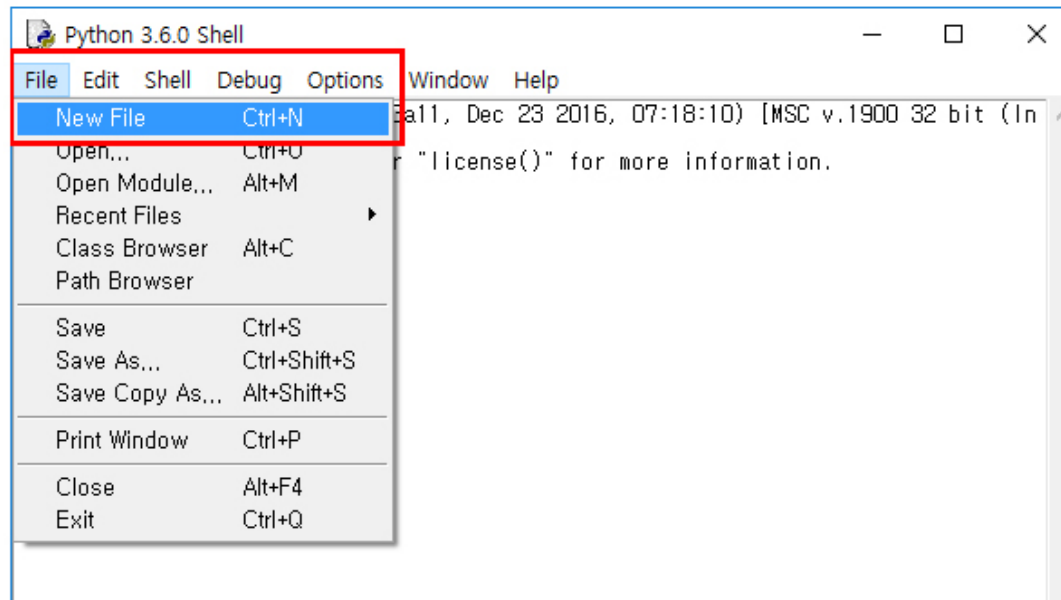


```
*Python 3.6.0 Shell*
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print(Hello, world!)
SyntaxError: invalid syntax
>>> print(Hello, world!)
```

IDLE 파이썬 셸에서 이전 코드 사용하기

- **IDLE에서 소스 파일 실행하기**

- `print('Hello, world!')` 코드를 파일에 저장해서 실행
- 프로그래밍에서 코드를 파일 형태로 저장한 것을 소스 코드 또는 소스 파일
- IDLE은 파이썬 셸을 통해 코드를 실행하고 바로 바로 결과를 보는 기능과 소스 파일을 편집하고 실행하는 기능을 함께 제공
- IDLE을 실행하고 메뉴에서 File > New File을 클릭



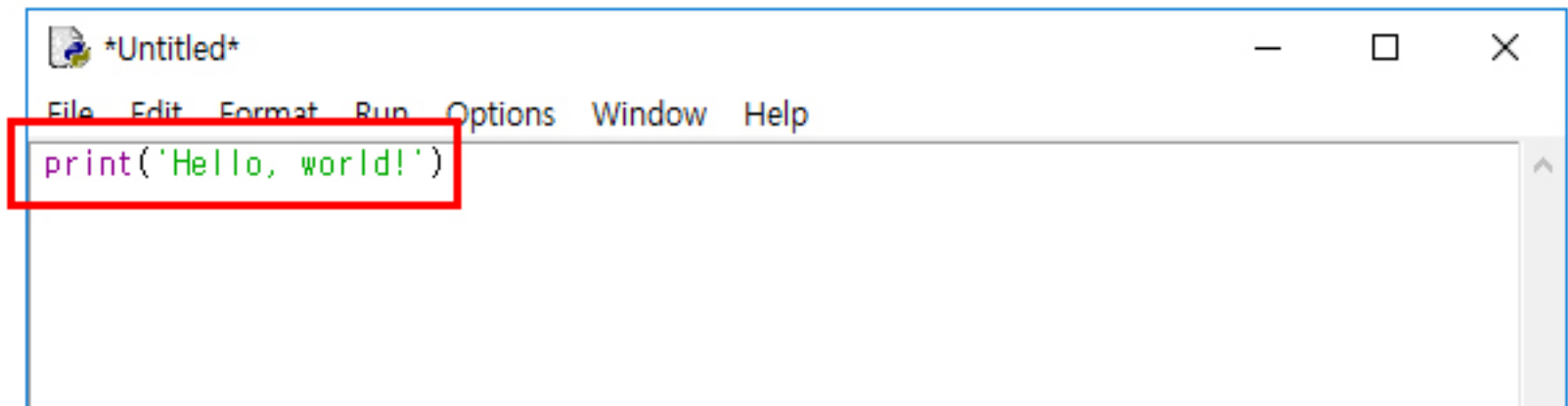
IDLE에서 New File 클릭

- IDLE에서 소스 파일 실행하기

- 내용이 비어 있는 소스 코드 편집창이 나옴
- 다음 내용을 소스 코드 편집 창에 입력

hello.py

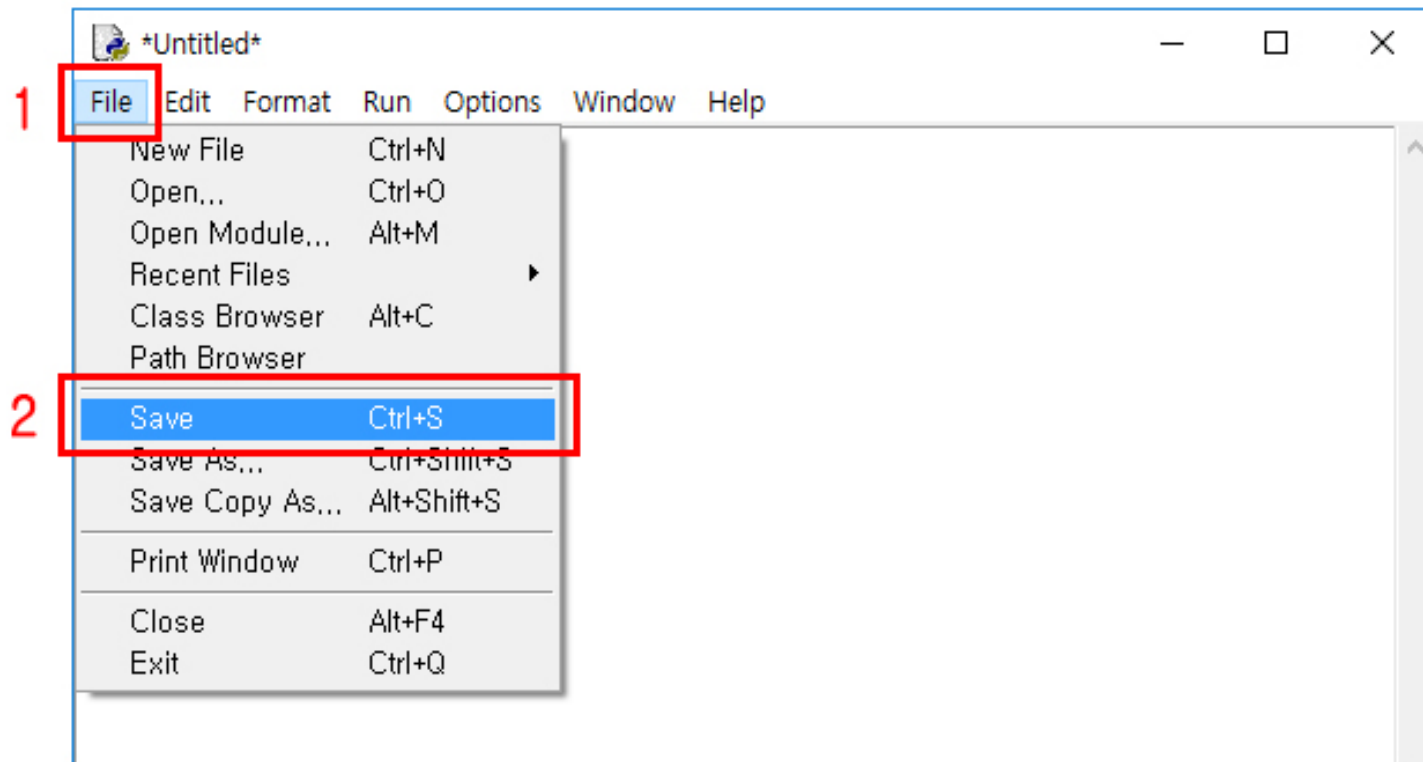
```
print('Hello, world!')
```



파이썬 소스 코드 입력

- IDLE에서 소스 파일 실행하기

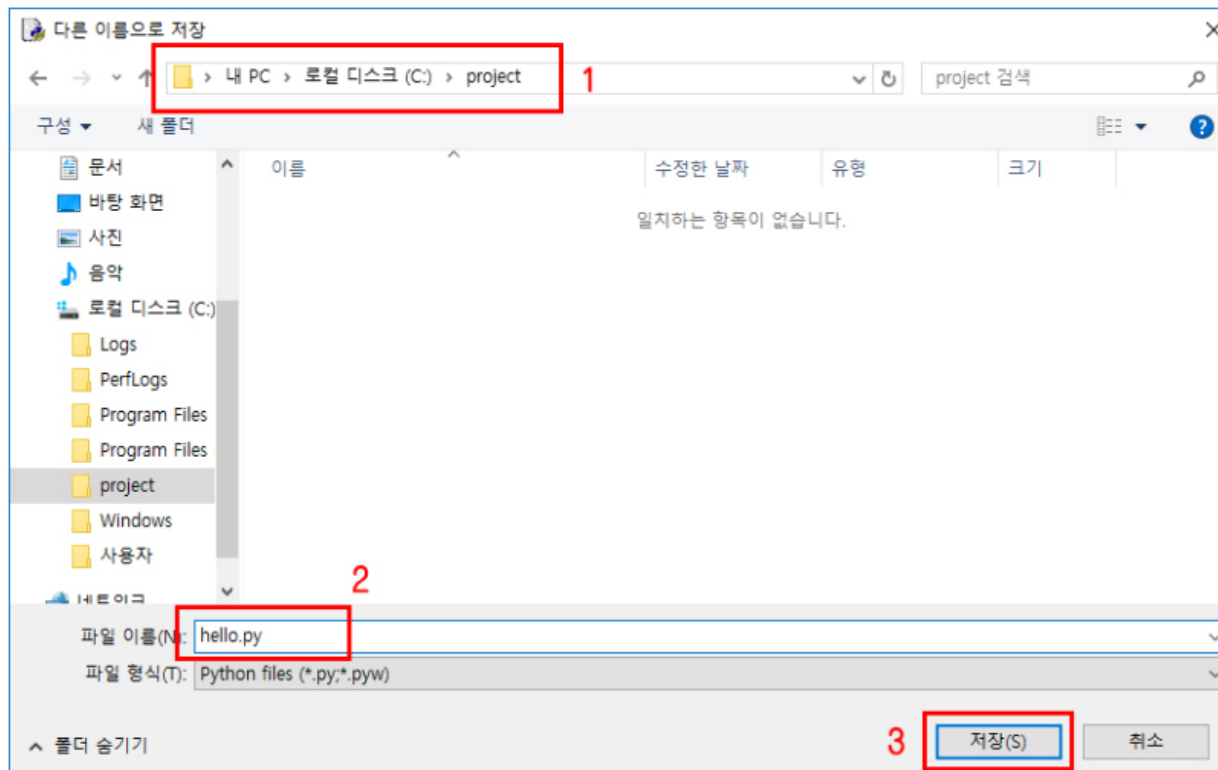
- 입력한 파이썬 코드를 실행하려면 .py 파일에 저장
- 소스 코드 편집 창의 메뉴에서 File > Save를 클릭하거나 Ctrl+S를 누름



파이썬 코드를 파일에 저장

- IDLE에서 소스 파일 실행하기

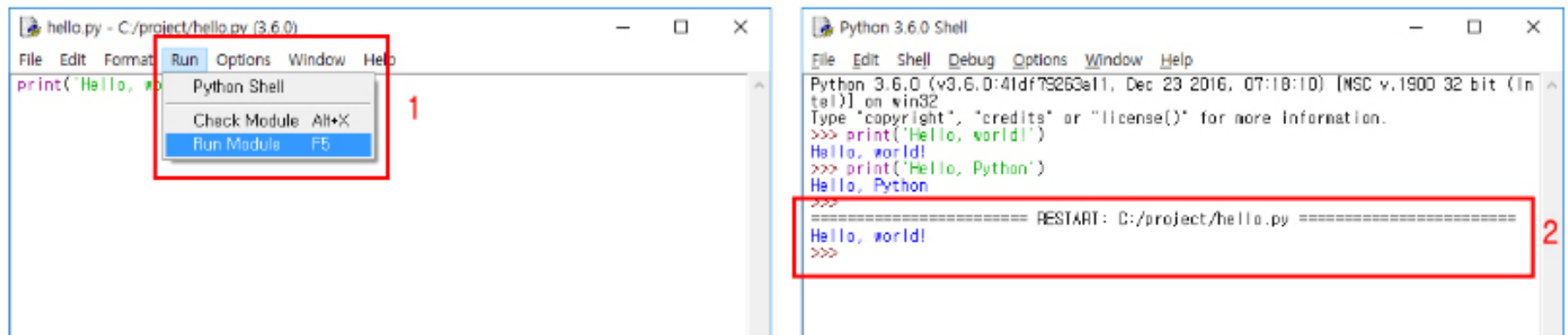
- 파일 저장 창이 표시되면 C:\wproject 폴더로 이동한 뒤 hello.py로 저장



C:\wproject에 hello.py로 저장

- IDLE에서 소스 파일 실행하기

- 파이썬 코드를 hello.py 파일로 저장
- 소스 코드 편집 창의 메뉴에서 Run > Run Module을 클릭하거나 F5 키를 누르면 IDLE의 파이썬 셸 창에 Hello, world!가 출력
- IDLE의 소스 코드 편집 창에 내용을 입력한 뒤 실행하면 파이썬 셸 창에 결과가 출력
- 각자 print() 안의 내용을 바꿔서 실행
- 파이썬 코드를 저장한 .py 파일을 파이썬 스크립트(Python script)



파이썬 소스 파일 실행

- 소스 코드 살펴보기

- 파이썬을 모르는 상태에서도 얼핏 보면 print가 'Hello, world!'를 출력한다는 것을 알 수 있음
- '(작은따옴표)로 묶은 부분을 문자열이라 하고, print는 값을 화면에 출력

hello.py

```
print ('Hello, world!')
```

- print처럼 단어 뒤에 ()(괄호)가 붙은 것을 함수(function)라고 하며 정해진 일을 수행하는 단위
- 함수는 print('Hello, world!')와 같이 함수 이름 print를 써주고, 괄호 안에 출력할 내용을 넣으면 함수가 실행
- 함수 실행을 다른 말로는 함수를 호출(call)한다고 말하기도 함

- 소스 코드 살펴보기

- 파이썬을 모르는 상태에서도 얼핏 보면 print가 'Hello, world!'를 출력한다는 것을 알 수 있음
- '(작은따옴표)로 묶은 부분을 문자열이라 하고, print는 값을 화면에 출력

hello.py

```
print ('Hello, world!')
```

- print처럼 단어 뒤에 ()(괄호)가 붙은 것을 함수(function)라고 하며 정해진 일을 수행하는 단위
- 함수는 print('Hello, world!')와 같이 함수 이름 print를 써주고, 괄호 안에 출력할 내용을 넣으면 함수가 실행
- 함수 실행을 다른 말로는 함수를 호출(call)한다고 말하기도 함

- **세미콜론**

- 파이썬은 세미콜론을 붙이지 않음

파이썬

```
print('Hello, world!')
```

- 세미콜론을 붙여도 문법 에러는 발생하지 않음
- 보통 한 줄에 여러 구문을 사용할 때 세미콜론으로 구분해줌

파이썬

```
print('Hello'); print('1234')
```

- **주석**

- 파이썬에서 사람만 알아볼 수 있도록 작성하는 부분을 주석(comment)라고 함
- 주석은 파이썬 인터프리터가 처리하지 않으므로 프로그램의 실행에는 영향을 주지 않음
- 보통 주석은 코드에 대한 자세한 설명을 작성하거나, 특정 코드를 임시로 사용하지 않도록 만들 때 사용
- 주석은 한 줄 주석과 블록 주석 두 가지가 있음

- 한 줄 주석

- 한 줄 주석으로 코드에 대한 설명을 작성한 모습

```
# Hello, world! 출력  
print('Hello, world!')
```

- 코드 맨 앞에 #을 사용하면 해당 줄은 모두 주석이 됨
- 다음 print 함수는 동작하지 않음

```
#print('Hello, world!')
```

- 코드 뒤에 #으로 주석을 작성할 수도 있음
- 이때는 앞에 있는 코드만 정상적으로 동작하며 # 뒤에 있는 코드는 동작하지 않음

```
a = 1 + 2 # 더하기  
print('Hello, world!') #printf('1234567890')
```


- **블록 주석**

- 블록 주석은 각 줄마다 맨 앞에 #을 넣어줌

```
#print('Hello, world!')  
#print('1234567890')
```

- 보통 블록 주석을 작성할 때는 코드를 읽기 쉽도록 # 뒤에 공백을 한 칸 띄움

```
# 더하기  
# a = 1 + 2  
# print('Hello, world!')
```

- cf) 파이썬에서 한글 주석 사용하기

- 파이썬 3에서는 .py 스크립트 파일의 기본 인코딩이 UTF-8
- 스크립트 파일을 다른 인코딩(CP949, EUC-KR)으로 저장하면 실행을 했을 때 에러가 발생

CP949로 저장된 hello.py

```
print('Hello, world!')    # 한글 주석
```

File "hello.py", line 1

SyntaxError: Non-UTF-8 code starting with '\xc7' in file hello.py on line 1, but no encoding declared; see <http://python.org/dev/peps/pep-0263/> for details

- 이때는 스크립트 파일을 UTF-8로 저장
- 대부분의 텍스트 편집기는 저장할 파일의 인코딩을 설정할 수 있음
- 메모장에서는 파일(F) > 다른 이름으로 저장(A)... > 인코딩(E)에서 UTF-8을 선택한 뒤 저장

- 들여쓰기

- 들여쓰기는 코드를 읽기 쉽도록 일정한 간격을 띄워서 작성하는 방법
- 특히 파이썬은 들여쓰기 자체가 문법
- 예를 들어 if의 다음 줄은 항상 들여쓰기를 해야 함
- 만약 들여쓰기를 하지 않으면 문법 에러이므로 코드가 실행되지 않음

파이썬

```
if a == 10:  
print('10입니다.')    # 들여쓰기 문법 에러
```

실행 결과

```
IndentationError: expected an indented block
```

파이썬

```
if a == 10:  
    print('10입니다.')
```

- 들여쓰기

- 파이썬에서 들여쓰기 방법은 공백(스페이스) 2칸, 4칸, 탭(tab) 등 여러 가지 방법이 있음
- 파이썬은 공백 2칸, 공백 4칸, 탭 문자 등을 각각 사용해도 동작이 잘 됨
- **코딩 스타일 가이드(PEP 8)**에서는 **공백 4칸으로 규정**하고 있음
- 공백 4칸을 사용하는 것이 좋음

공백 2칸 if a == 10:
 └─┘print('10입니다.')

공백 4칸 if a == 10:
 └─└─└─┘print('10입니다.')

탭 1칸 if a == 10:
 └─────────┘print('10입니다.')

들여쓰기 방법

• 코드 블록

- 코드 블록은 특정한 동작을 위해서 코드가 모여 있는 상태
- 파이썬은 들여쓰기를 기준으로 코드 블록을 구성

파이썬

```
if a == 10:
    print('10')
    print('입니다.')
```

- 단, 같은 블록은 들여쓰기 칸 수가 같아야 하고, 공백과 탭 문자를 섞어 쓰면 안 됨

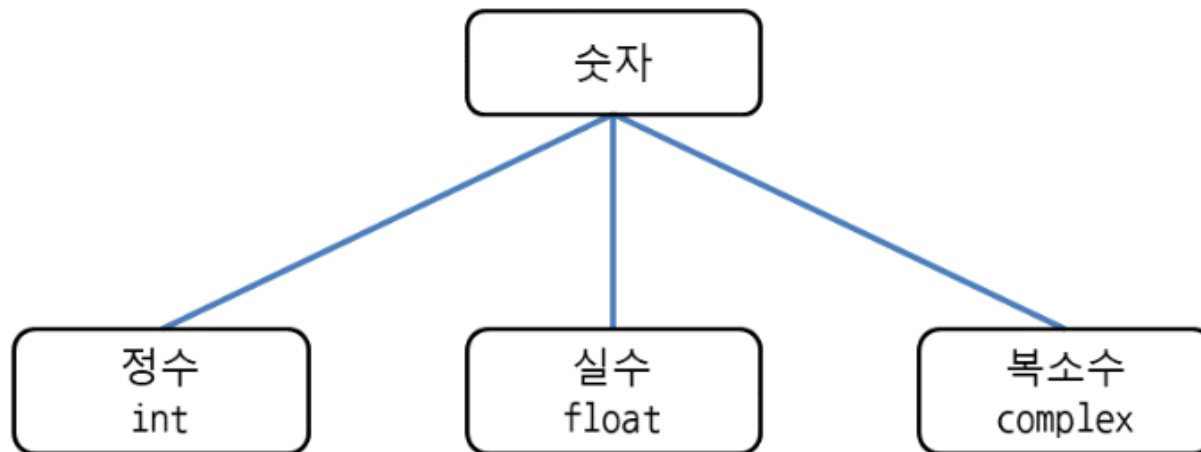
	if a == 10:	
공백 2칸	__print('10')	✗
공백 4칸	____print('입니다.')	

	if a == 10:	
공백 4칸	____print('10')	✗
탭 1칸	————>print('입니다.')	

	if a == 10:	
공백 4칸	____print('10')	◯
공백 4칸	____print('입니다.')	

- 숫자 계산하기

- 파이썬을 계산기처럼 사용해보기
- 숫자 계산을 하기 전에 먼저 숫자의 자료형(타입)부터 살펴보자
- 파이썬에서는 숫자의 자료형에 따라 결과가 달라질 수 있으므로 이 부분은 정확하게 구분할 필요가 있음
- 다음과 같이 파이썬에서는 숫자를 정수, 실수, 복소수로 구분



파이썬의 숫자 자료형

- 숫자 계산하기

- 보통 프로그래밍에서는 정수와 실수를 주로 사용하며 복소수는 공학 분야에서 주로 쓰임

- 정수 계산하기

- 파이썬 IDLE를 실행하거나 콘솔(터미널, 명령 프롬프트)에서 파이썬을 실행

```
C:\Users\dojang>python
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 사칙연산

- 파이썬 프롬프트가 나오면 코드와 계산식을 입력받을 준비가 된 상태
- >>>에 1 + 1을 입력한 뒤 엔터 키를 누르면 결과값 2가 나옴

```
>>> 1 + 1
2
```

- 덧셈뿐만 아니라 뺄셈, 곱셈도 가능

```
>>> 1 - 2
-1
>>> 2 * 2
4
```

- 나눗셈

```
>>> 5 / 2
2.5
```


- 사칙연산

- 이번에는 나눗셈

```
>>> 5 / 2  
2.5
```

- 5 나누기 2는 2.5가 나옴
- 당연한 결과일 수도 있지만 이 부분이 파이썬 2와 파이썬 3의 차이점
- 파이썬 2에서 $5 / 2$ 는 2.5가 아닌 2가 나오는데, 정수끼리 나눗셈 결과는 정수가 나오도록 정했기 때문임
- 파이썬 3는 정수끼리 나눗셈을 해도 실수가 나옴
- 4 나누기 2를 계산해보면 정말 그런지 확인할 수 있음

```
>>> 4 / 2  
2.0
```

- 파이썬 3은 나눗셈이 완전히 나누어 떨어져도 실수가 나옴

- 나눗셈 후 소수점 이하를 버리는 // 연산자

- 파이썬 3에서 정수끼리 나눗셈 결과가 정수로 나도록 만들어 보자
- //로 나눗셈을 하면 됨

```
>>> 5 // 2
2
>>> 4 // 2
2
```

- //은 버림 나눗셈(floor division)이라고 부르며 나눗셈의 결과에서 소수점 이하는 버림
- 실수에 // 연산자를 사용하면 결과는 실수가 나오며 소수점 이하는 버림
- 결과는 항상 .0으로 끝남

```
>>> 5.5 // 2
2.0
>>> 4 // 2.0
2.0
>>> 4.1 // 2.1
1.0
```

- 나눗셈 후 나머지를 구하는 % 연산자

- 나눗셈 후 나누어 떨어지지 않을 때 나머지를 구해보자

```
>>> 5 % 2  
1
```

- 5를 2로 나누면 두 번 나눌 수 있고 1이 남음
- %는 두 수를 나누었을 때 나머지만 구하며 모듈로(modulo) 연산자라고 부름
- 몫은 버림 나눗셈(/)으로 구할 수 있음

- 거듭제곱을 구하는 ** 연산자

- 파이썬에서는 거듭제곱도 쉽게 구할 수 있음

```
>>> 2 ** 10  
1024
```

- **은 거듭제곱 연산자이며 숫자를 특정 횟수만큼 곱함
- $2^{**}10$ 은 1024이고 2^{10} 을 뜻함

- **값을 정수로 만들기**

- 계산 결과가 실수로 나왔을 때 강제로 정수로 만들어보기
- **int에 괄호를 붙이고 숫자 또는 계산식을 넣으면 됨**
- int에 문자열을 넣어도 정수로 만들 수 있음
- 단, 정수로 된 문자열이어야 함

- int(숫자)
- int(계산식)
- int('문자열')

```
>>> int(3.3)
3
>>> int(5 / 2)
2
>>> int('10')
10
```

- int는 정수(integer)를 뜻하며 값을 정수로 만들어 줌(소수점 이하는 버림)

- 객체의 자료형 알아내기

- 지금까지 사용한 숫자가 정수가 맞는지 확인해보자

- `type(값)`

```
>>> type(10)
<class 'int'>
```

- **type**은 말그대로 객체의 타입(자료형)을 알아내는 함수
- **type**에 괄호를 붙이고 10을 넣어보면 <class 'int'>라고 나오는데 정수(int) 클래스라는 뜻
- 파이썬에서는 숫자도 객체(object)이며, 객체는 클래스(class)로 표현함
- 참고로 앞에서 사용한 `int(3.3)`은 실수 3.3을 int 클래스로 된 객체로 만든다는 뜻

- cf) 몫과 나머지를 함께 구하기

- 몫과 나머지를 함께 구하려면 **divmod**를 사용

```
>>> divmod(5, 2)
(2, 1)
```

- 지금까지 계산 결과가 1, 2, 1024처럼 값만 나왔지만 divmod는 결과가 (2, 1)처럼 나옴
- 파이썬에서 값을 괄호로 묶은 형태를 튜플(tuple)이라고 하며 값 여러 개를 모아서 표현할 때 사용
- 튜플은 차후 자세히 설명하겠음
- 튜플은 변수 여러 개에 저장할 수 있는데 divmod의 결과가 튜플로 나오므로 몫과 나머지는 변수 두 개에 저장할 수 있음

```
>>> quotient, remainder = divmod(5, 2)
>>> print(quotient, remainder)
2 1
```

- **cf) 진수, 8진수, 16진수**

- 정수는 10진수 이외에도 2진수, 8진수, 16진수로도 표현할 수 있음
- 2진수: 숫자 앞에 0b를 붙이며 0과 1을 사용
- 8진수: 숫자 앞에 0o(숫자 0과 소문자 o)를 붙이며 0부터 7까지 사용
- 16진수: 숫자 앞에 0x 또는 0X를 붙이며 0부터 9, A부터 F까지 사용
(소문자 a부터 f도 가능)

```
>>> 0b110
6
>>> 0o10
8
>>> 0xF
15
```

- 실수 계산하기

- 실수끼리 계산해보자
- >>>에 3.5 + 2.1를 입력해보자

```
>>> 3.5 + 2.1  
5.6
```

- 덧셈의 결과인 5.6이 잘 나왔음
- 이번에는 뺄셈, 곱셈, 나눗셈을 해보자

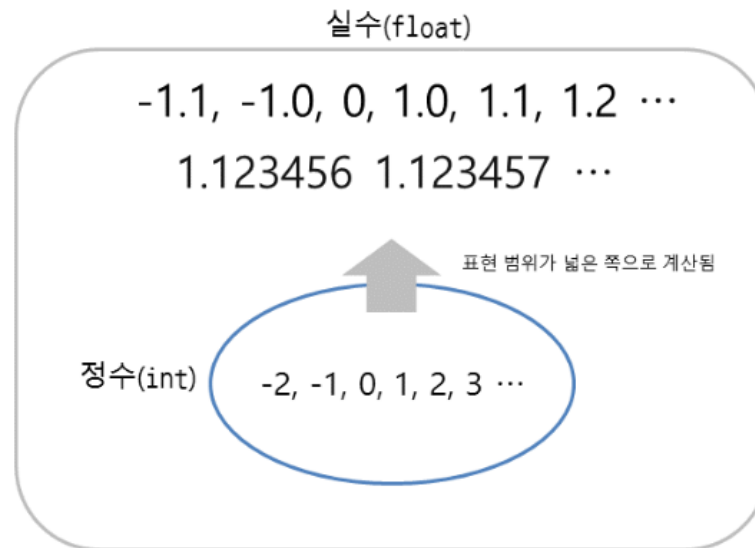
```
>>> 4.3 - 2.7  
1.5999999999999996  
>>> 1.5 * 3.1  
4.65  
>>> 5.5 / 3.1  
1.7741935483870968
```

- 오차 문제는 다소 어려운 주제이므로 나중에 실무에서 실수를 다룰 때 오차에 대한 적절한 처리가 필요함

- 실수와 정수를 함께 계산하면?
 - 실수와 정수를 함께 계산해보자

```
>>> 4.2 + 5  
9.2
```

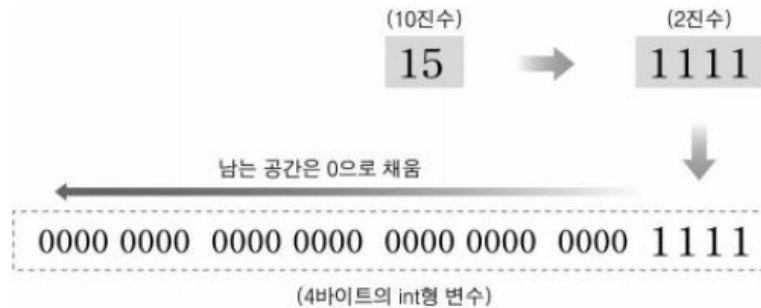
- 실수인 9.2가 나옴
- 실수와 정수를 함께 계산하면 표현 범위가 넓은 실수로 계산됨(실수가 정수보다 표현 범위가 넓음)



정수와 실수의 표현방식

정수의 표현방식

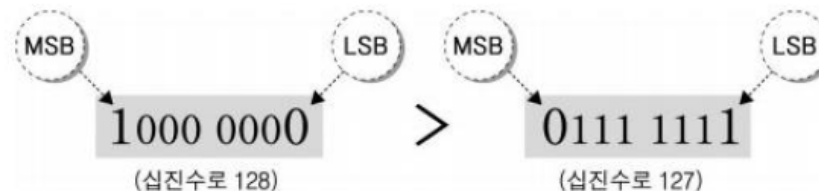
정수 값의 비트 표현



컴퓨터에서 최대 저장할 수 있는 범위를 넘어서는 경우 나머지 값만이 저장된다. (**OVERFLOW**)



2진수 비트열에서 가장 왼쪽의 비트는 MSB(Most Significant Bit),
가장 오른쪽의 비트는 LSB(Least Significant Bit)



정수의 음수와 양수를 모두 표현하기 위해 MSB부분을 부호(sign)비트로 사용함

unsigned int -> 양수만을 표현, **(signed) int** -> 음수와 양수까지 표현

unsigned char -> 양수만을 표현, **(signed) char** -> 음수와 양수까지 표현

: MSB는 부호비트로 0이면 양수, 1이면 음수

□ 직관적인 방법으로 음수를 표현할 때,

signed char (1byte)를 이용한 수의 표현 1)

비트표현(2진수형태)	10진수	비트표현(2진수형태)	10진수
0 000 0000	0	1 000 0000	- 0
0 000 0001	1	1 000 0001	- 1
0 000 0010	2	1 000 0010	- 2
0 000 0011	3	1 000 0011	- 3
⋮	⋮	⋮	⋮
0 111 1110	126	1 111 1110	-126
0 111 1111	127	1 111 1111	-127

- 부호비트를 항상 떼고 다시 연산해야 하는 번거로움이 있다.

- 0이 2개의 경우를 차지, 메모리를 중복 사용한다.

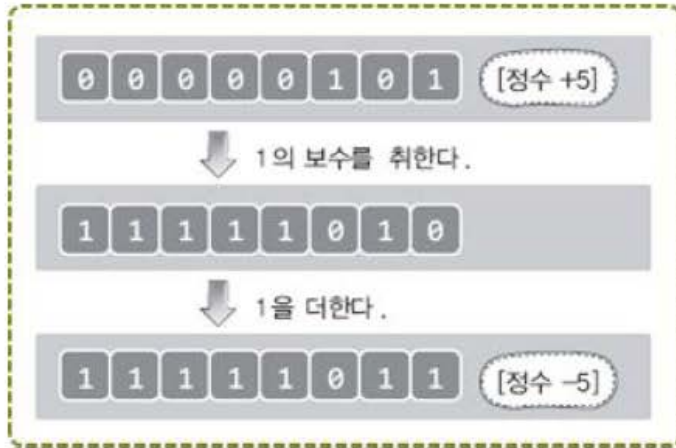
1의 보수의 경우는 1을 0으로 바꾸거나, 0을 1로 바꾸어서 얻는다.(반전시킨다.)

2의 보수의 경우는 반전시킨 다음, 1을 더해서 얻는다. = 1의 보수에 1을 더하여 구한다.

□ 1의 보수, 2의 보수를 이용하여 1byte 정수의 음수와 양수를 표현했을 때,
signed char (1byte)를 이용한 수의 표현 2)

비트표현 (2진수형태)	10진수			비트표현 (2진수형태)	10진수		
	1의 보수 (양수일때 사용안함)	2의 보수 (양수일때 사용안함)	unsigned		1의 보수	2의 보수	unsigned
0000 0000	0	0	0	1111 1111	- 0	- 1	255
0000 0001	1	1	1	1111 1110	- 1	- 2	254
0000 0010	2	2	2	1111 1101	- 2	- 3	253
0000 0011	3	3	3	1111 1100	- 3	- 4	252
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0111 1110	126	126	126	1000 0001	-126	-127	129
0111 1111	127	127	127	1000 0000	-127	-128	128

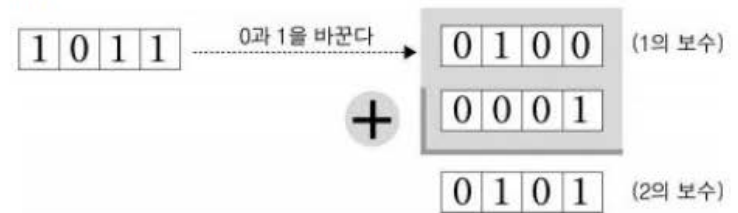
- 1의 보수로 표현하면 메모리의 중복은 여전히 피할 수 없다.
- 2의 보수를 컴퓨터에서 메모리의 중복을 피하여 음수를 표현하기 위해 사용한다.



2의 보수를 사용하여 음의 정수를 표현한다.

예제. 부호비트가 있는 4bits 이진수 1011의 10진수 값은 ?

- ① msb가 1이므로 음수이다.
- ② 2의 보수로 변경하면 0101 → 5
- ③ 1011의 십진수 값은 -5이다.



2의 보수를 사용하면, 뺄셈이 덧셈이 된다.



실수의 표현방식

부동 소수점 방식(floating-point system)

소수점 위치를 고정하지 않고 동일한 유효숫자 개수를 사용하는 지수 형식
한정된 메모리로 표현할 수 있는 표현 범위가 상대적으로 크다.
컴퓨터에 실수값은 IEEE754 표준에 따라 저장된다.

자료형	크기
float	4 byte
double	8 byte

32bit 컴퓨터 기준

배정밀도의 경우 저장되는 수의 형태는

$$x = (-1)^s \times (1.a)_2 \times (2^{e-1023})_{10} \text{ 이다.}$$

C언어에서 double 형 변수(8byte)에 실수 값을 저장하는 예

- ① MSB는 부호비트이며 0은 양수, 1은 음수이다.
- ② 부호비트 다음부터 11비트 (float는 8bits)는 지수값을 저장한다.
- ③ 나머지 52비트 (float는 23bits)는 소수값을 저장한다.



IEEE754 배정밀도(64bits, double) 표현

실수형 변수타입은 넓은 범위의 실수를 표현할 수 있지만, 실수의 표현에 오차가 존재한다. 이론적으로 오차 없이 모든 실수를 완벽하게 표현할 능력이 있는 컴퓨팅 환경은 존재하지 않는다. 즉, 실수 표현에 있어서의 오차 발생은 C언어의 특성이 아닌 컴퓨터의 기본적인 특성이다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i;
```

```
    float num=0.0;
```

```
    double dNum=0.0;
```

```
    for(i=0; i<100; i++)
```

```
    {
```

```
        num+=0.1;
```

```
        dNum += 0.1;
```

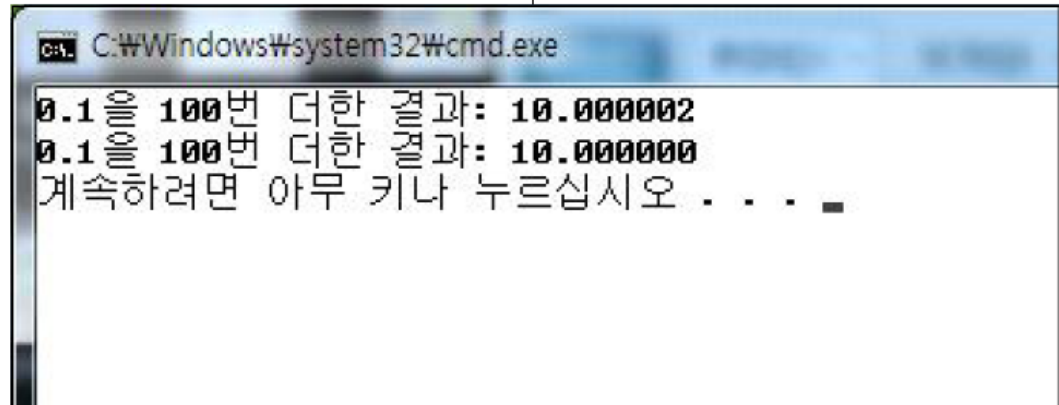
```
    }
```

```
    printf("0.1을 100번 더한 결과: %f \n", num);
```

```
    printf("0.1을 100번 더한 결과: %lf \n", dNum);
```

```
    return 0;
```

```
}
```



```
C:\Windows\system32\cmd.exe
0.1을 100번 더한 결과: 10.000002
0.1을 100번 더한 결과: 10.000000
계속하려면 아무 키나 누르십시오 . . .
```

- **값을 실수로 만들기**

- 숫자 또는 계산 결과를 강제로 실수로 만들려면?
- float에 괄호를 붙이고 숫자 또는 계산식을 넣으면 됨
- float에 문자열을 넣어도 실수로 만들 수 있음
- 단, 실수 또는 정수로 된 문자열이어야 함

- float(숫자)
- float(계산식)
- float('문자열')

```
>>> float(5)
5.0
>>> float(1 + 2)
3.0
>>> float('5.3')
5.3
```


- **값을 실수로 만들기**

- float는 부동소수점(floating point)에서 따왔으며 값을 실수로 만들어줌
- 즉, 실수는 float 자료형이며 type에 실수를 넣어보면 <class 'float'>가 나옴

```
>>> type(3.5)
<class 'float'>
```

- **cf) 복소수**

- 파이썬에서는 실수부와 허수부로 이루어진 복소수(complex number)도 사용할 수 있음
- 이때 허수부는 숫자 뒤에 j를 붙임(수학에서는 허수를 i로 표현하지만 공학에서는 j를 사용)

```
>>> 1.2+1.3j
(1.2+1.3j)
```

- 두 실수를 복소수로 만들 때는 complex를 사용하면 됨

```
>>> complex(1.2, 1.3)
(1.2+1.3j)
```

- cf) 스크립트 파일에서 계산 결과가 출력되지 않아요

- 스크립트 파일에서 $1 + 1$ 처럼 계산식만 넣으면 결과가 출력되지 않음

print_add.py

```
1 + 1
```

실행 결과

(아무것도 출력되지 않음)

- 스크립트 파일에서 계산 결과를 출력하려면 print 함수를 사용해야 함

print_add.py

```
print(1 + 1)
```

실행 결과

2

- 파이썬 셸은 결과를 즉시 보는 용도라서 값 또는 계산식만 넣어도 결과가 출력됨
- 스크립트 파일에서 값 또는 계산식의 결과를 출력하려면 반드시 print 함수를 사용해야 함

- 괄호 사용하기

- $35 + 1 * 2$ 의 결과는 무엇일까요?
- 식에서는 덧셈, 곱셈 순으로 나와있지만 곱셈을 덧셈보다 먼저 계산하므로 72가 아닌 37이 정답임
- 파이썬에서도 마찬가지임

```
>>> 35 + 1 * 2
37
```

- 곱셈보다 덧셈을 먼저 계산하고 싶다면 덧셈 부분을 괄호로 묶어주면 됨

```
>>> (35 + 1) * 2
72
```

- 덧셈, 뺄셈, 곱셈, 나눗셈이 함께 있을 때는 곱셈과 나눗셈부터 계산
- 덧셈과 뺄셈을 먼저 계산하고 싶다면 괄호로 묶어줌
- 프로그램을 만들 때 자주 사용하니 꼭 기억해두자

우선순위	연산자	설명
1	(값...), [값...], {키: 값...}, {값...}	튜플, 리스트, 딕셔너리 세트 생성
2	x[인덱스], x[인덱스:인덱스], x(인수...), x.속성	리스트(튜플) 첨자, 슬라이싱, 함수 호출, 속성 참조
3	await x	await 표현식
4	**	거듭제곱
5	+x, -x, ~x	단항 덧셈(양의 부호), 단항 뺄셈(음의 부호), 비트 NOT
6	*, @, /, //, %	곱셈, 행렬 곱셈, 나눗셈, 버림 나눗셈, 나머지
7	+, -	덧셈, 뺄셈
8	<<, >>	비트 시프트
9	&	비트 AND
10	^	비트 XOR
11		비트 OR
12	in, not in, is, is not, <, <=, >, >=, !=, ==	포함 연산자, 객체 비교 연산자, 비교 연산자
13	not x	논리 NOT
14	and	논리 AND
15	or	논리 OR
16	if else	조건부 표현식
17	lambda	람다 표현식

파이썬 연산자 우선순위

- **괄호 사용하기**

- 실생활에서는 1과 1.0을 구분하지 않지만 컴퓨터는 1과 1.0을 정수와 실수로 구분
- 파이썬에서도 정수와 실수는 구분해서 처리함
- 나눗셈 연산자 `/`의 결과는 실수, 버림 나눗셈 `//`의 결과는 정수라는 점을 기억