

PYTHON 기본 환경

동의대학교 컴퓨터소프트웨어공학과
이종민 교수

참고 문헌

- The Python Language Reference, <https://docs.python.org/3/reference/index.html> (2017년 6월).
- The Python Standard Library, <https://docs.python.org/3/library/index.html> (2017년 6월).
- The Python Tutorial, <https://docs.python.org/3/tutorial/index.html> (2017년 6월).
- Python HOWTO, <https://docs.python.org/3/howto/index.html> (2017년 6월)

- 폴 베리, 데이비드 그리피스 (강권학 역), Head First Programming, 한빛미디어.
- 에릭 프리먼 외 (서환수 역), Head First Design Patterns, 한빛미디어.

목차

- 개발 환경
- 함수
- 클래스
- 식별자
- 리터럴: 문자열, 정수, 부동소수

Python 3.x 설치

- Python 3.x – 개발 진행 버전
- Python 2.7.x – 유지보수 버전 (<https://pythonclock.org/>)
- 내려 받기 URL: <http://www.python.org/download/>
- 설치
 - 64비트 버전 설치해야 함.
 - 일부 패키지는 64비트 버전 지원 안 할 수도 있음.
 - 참고. Unofficial Windows Binaries for Python Extension Packages, <http://www.lfd.uci.edu/~gohlke/pythonlibs/> → pygame, opencv 등 (.whl 포맷 제공: Wheel Binary Package. PEP 427)

- [Latest Python 3 Release - Python 3.8.5](#)
- [Latest Python 2 Release - Python 2.7.18](#)

파이썬 버전 별 특징

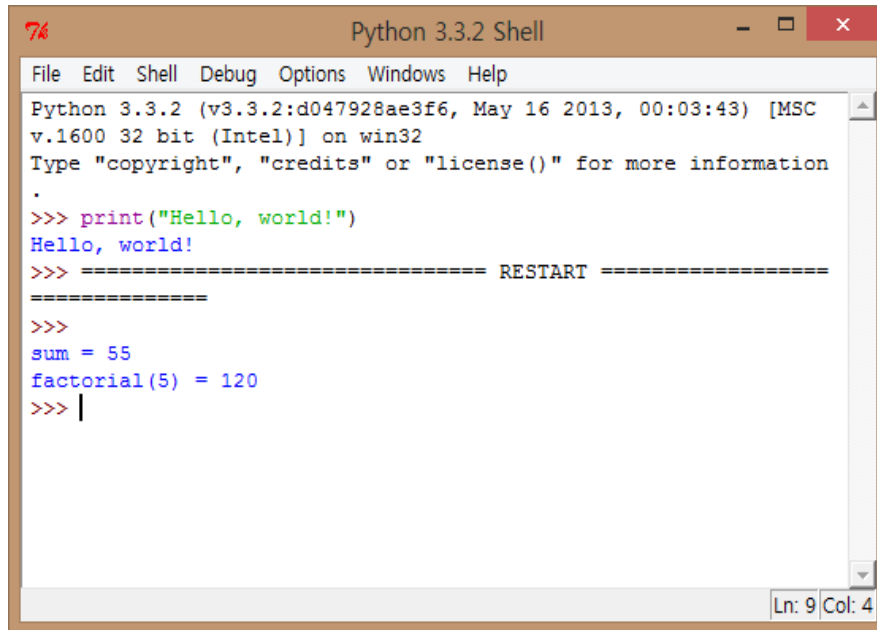
파이썬 버전 (릴리스 연월) ¹⁾	특징
3.7 (2018년 06월)	PEP 539 - 스레드-로컬 스토리지용 새 C API PEP 545 - Python 문서 번역. (일본어, 프랑스어, 한국어 추가) PEP 553 - 내장 breakpoint() 함수 PEP 557 - 데이터 클래스 PEP 564 - 나노초 해상도를 가지는 시간 함수 PEP 567 - 컨텍스트 변수
3.6 (2016년 12월)	PEP 498 - 형식화된 문자열 스트링 PEP 515 - 숫자 리터럴의 밑줄 PEP 526 - 변수 애노테이션 구문 PEP 525 - 비동기 발생기(generator) PEP 530 - 비동기 comprehension secrets 라이브러리 추가
3.5 (2015년 09월)	PEP 492 - async/await 구문을 가지는 코루틴 PEP 465 - 행렬 곱셈 연산자: a @ b PEP 448 - 추가적인 unpacking 일반화: *, ** PEP 484 - 자료형 힌트: typing PEP 441을 개선한 파이썬 ZIP 응용 지원

3.4 (2014년 03월)	추가된 라이브러리 모듈: asyncio, ensurepip, enum, pathlib, selectors, statistics, tracemalloc 보안 개선: 안전하고 교환가능한 해쉬 알고리즘(PEP 456), TLSv1.1/TLSv1.2 지원
3.3 (2012년 09월)	yield from 표현식 추가 라이브러리 모듈: faulthandler, ipaddress, lkm, unittest.mock, venv(파이썬 가상 환경 - PEP 405)
3.2 (2011년 09월)	unittest 모듈 개선 PEP 3147 - .pyc 저장소 디렉토리 지원 PEP 3148 - 병행 프로그래밍을 위한 concurrent.futures 모듈 PEP 391 - 딕셔너리-기반 로깅 환경 설정 configparser 개선 Python 디버거 pdb 개선

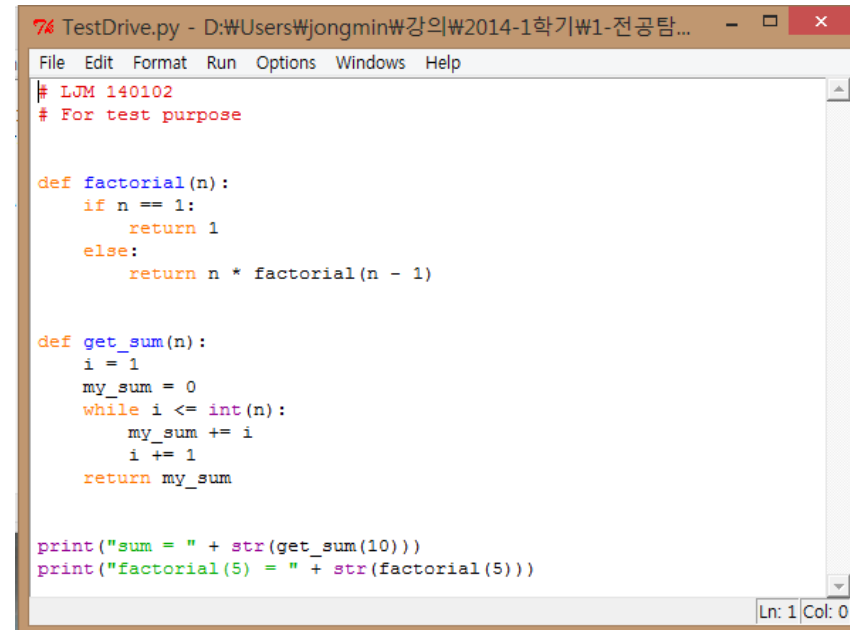
3.1 (2009년 06월)	<p>PEP 372 - 순서있는 덱서너리</p> <p>PEP 378 - 천단위 구분을 위한 형식 지정자: <code>format(1234567, ',d')</code></p> <p>테스트 스킵과 assert 메소드를 포함하는 unittest 모듈 추가</p> <p>빠른 io 모듈</p> <p>Tkinter용 Tile 지원</p>
3.0 (2008년 12월)	<p>print문을 print() 함수로 대체</p> <p>long 자료형을 int로 재명명 (기존의 long과 거의 유사, PEP 0237))</p> <p>sys.maxint 제거 & sys.maxsize 추가</p> <p>PEP 3120 - UTF-8을 기본 소스 인코딩으로 사용</p> <p>PEP 3131 - ASCII 문자가 아닌 문자도 식별자로 사용 가능</p> <p>PEP 3104 - nonlocal 문</p> <p>PEP 3132 - 확장된 iterable unpacking</p>

IDLE

- tkinter GUI 툴킷을 사용하여 만든 Python IDE
- 100% 순수 Python 코드로 개발
- Windows와 Unix 운영체제 호환
- 디버깅 가능 (breakpoint, view, step 기능 이용)



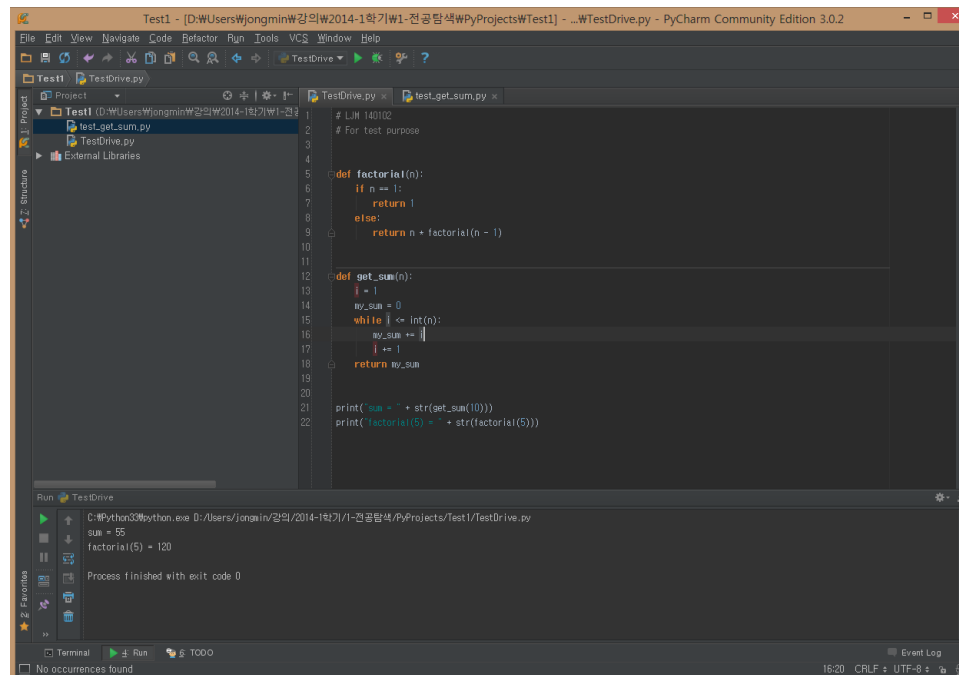
```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC  
v.1600 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information  
>>> print("Hello, world!")  
Hello, world!  
>>> ===== RESTART =====  
>>>  
sum = 55  
factorial(5) = 120  
>>> |
```



```
# LJM 140102  
# For test purpose  
  
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
def get_sum(n):  
    i = 1  
    my_sum = 0  
    while i <= int(n):  
        my_sum += i  
        i += 1  
    return my_sum  
  
print("sum = " + str(get_sum(10)))  
print("factorial(5) = " + str(factorial(5)))
```


PyCharm

- The intelligent Python IDE with unique code assistance and analysis, for productive Python development on all levels
- 내려 받기 URL: <http://www.jetbrains.com/pycharm/download/>
 - Professional Edition : 30일 무료
 - **Community Edition : 무료 (Python IDE로 사용)**



PyCharm 특징

- 코드 완성, 오류 하이라이팅, 자동 수정 등의 기능이 있는 편집기
- 자동화된 코드 리팩토링과 풍부한 네비게이션 기능
- 통합 디버거와 단위 테스트 지원
- Apache2 라이선스 적용

주의) PyCharm 설치하기 전에 JDK 설치되어 있어야 함.

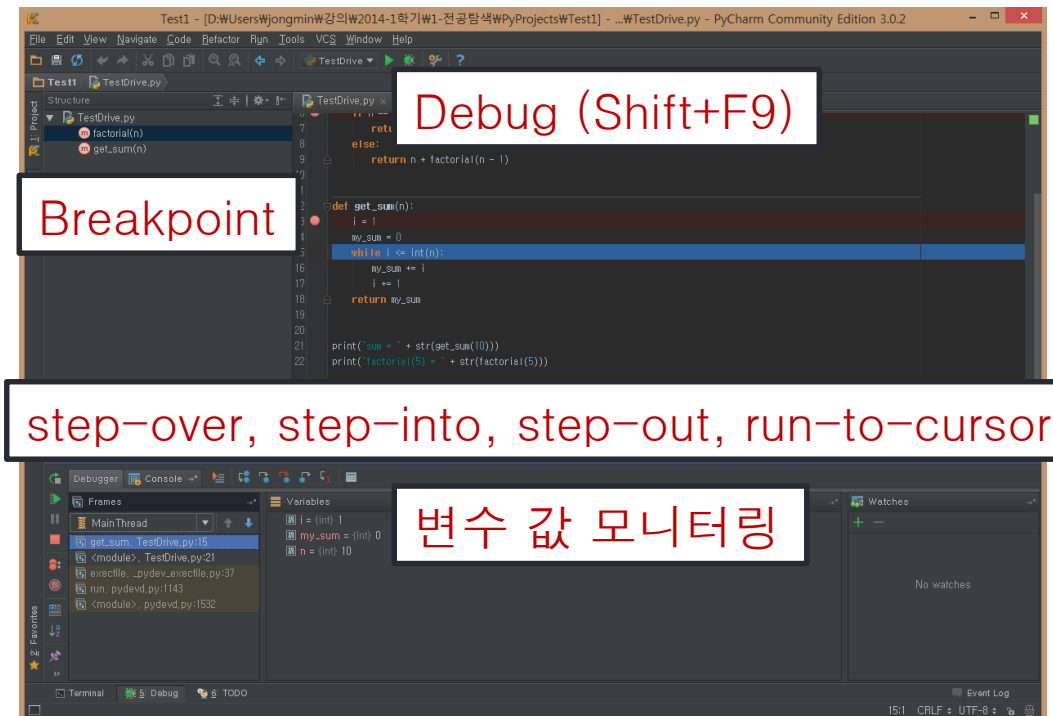
→ <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (google에서 "Oracle JDK download" 검색)

cf. JDK: Java Development Kit

디버깅 (Debugging)

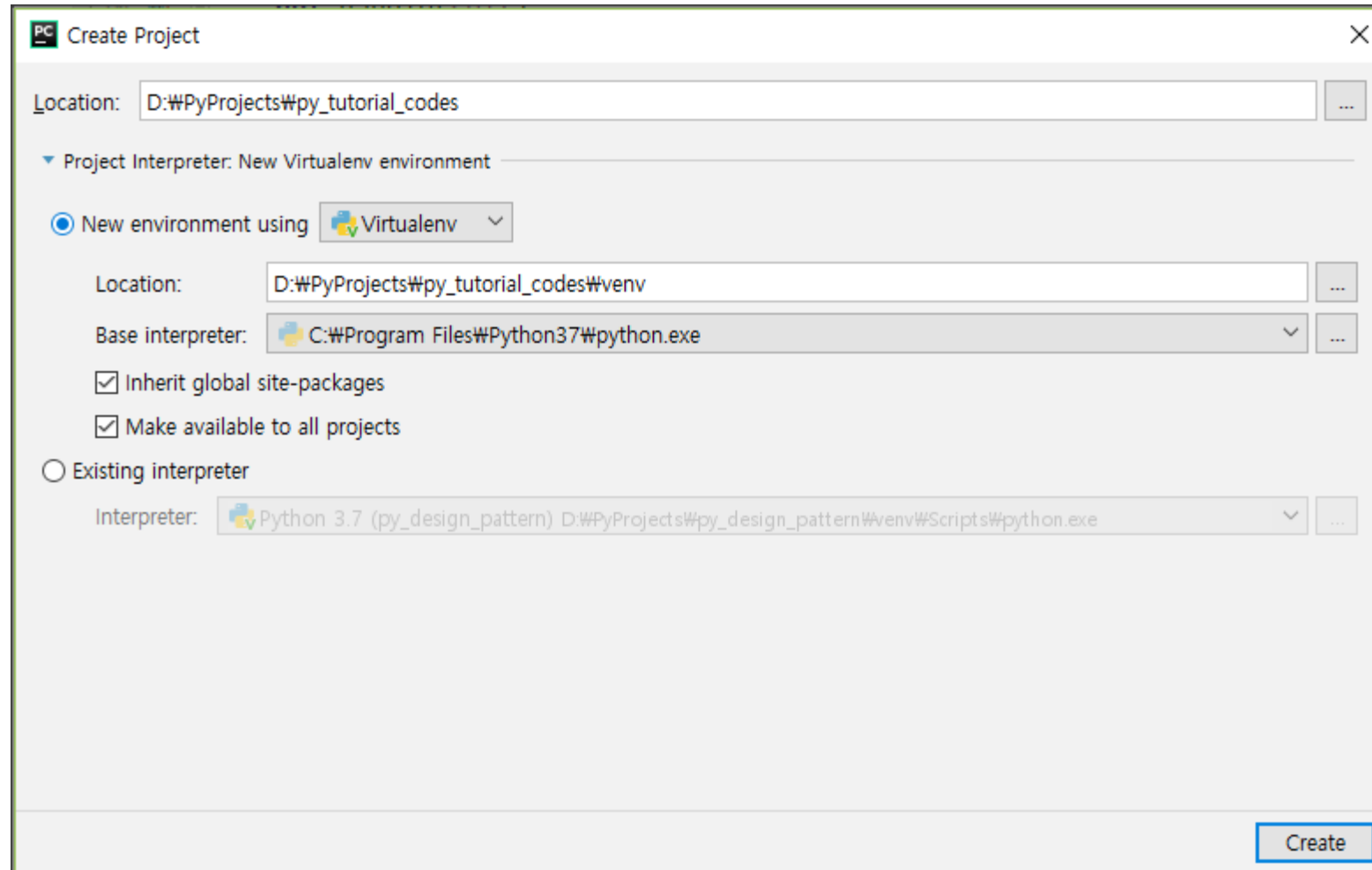
- 정의

- a methodical process of finding and reducing the number of bugs, or defects, in a computer program ... (from Wikipedia)
- 결함 (defect; 아이템 중에 존재하는 이상(규격에서 벗어난 것) 등 고장의 원인이 되는 상태 또는 장소. TTA 용어사전)을 찾아내고 수정하는 행위 또는 절차

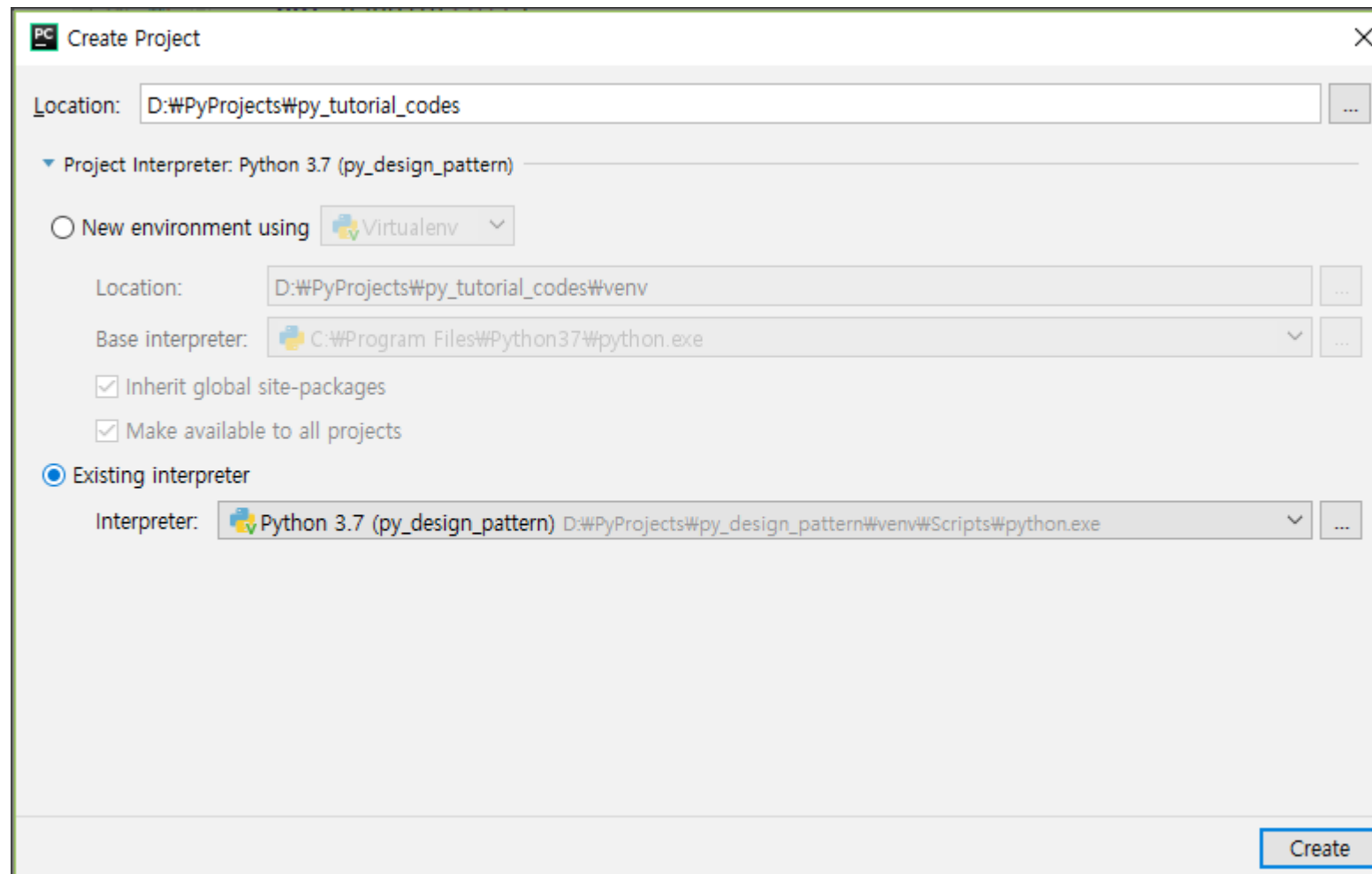


파이썬 프로젝트 생성

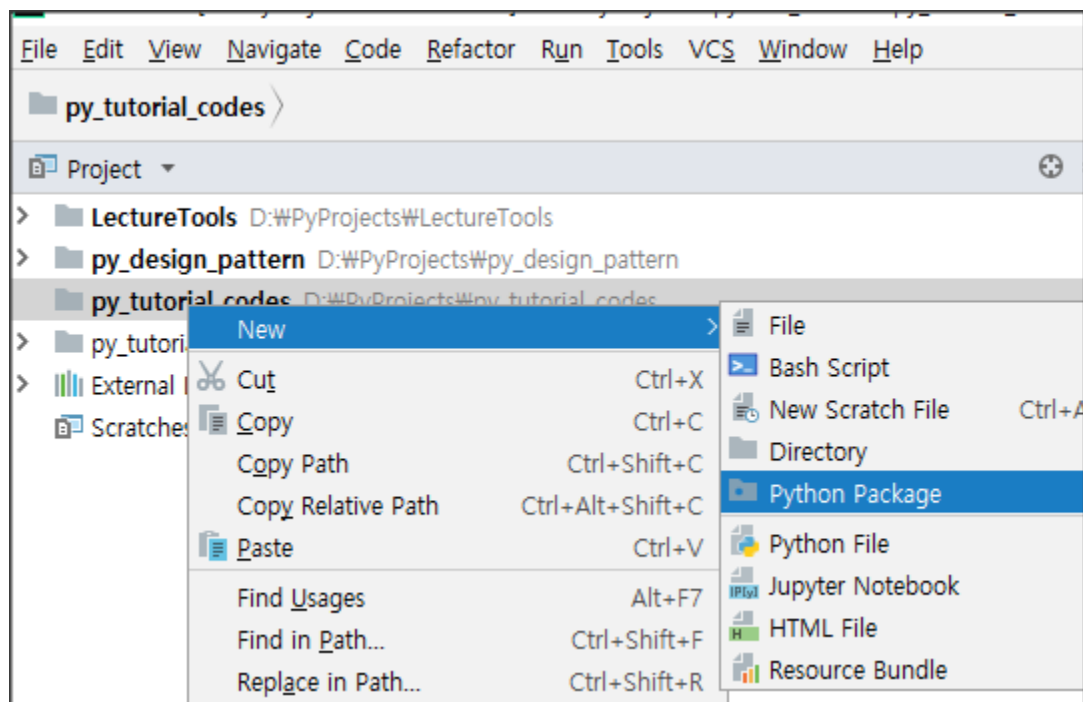
- (1) 가상 환경 사용 (선호) – 다른 개발자와의 패키지 충돌 예방 가능



- (2) 기존 파이썬 인터프리터 사용

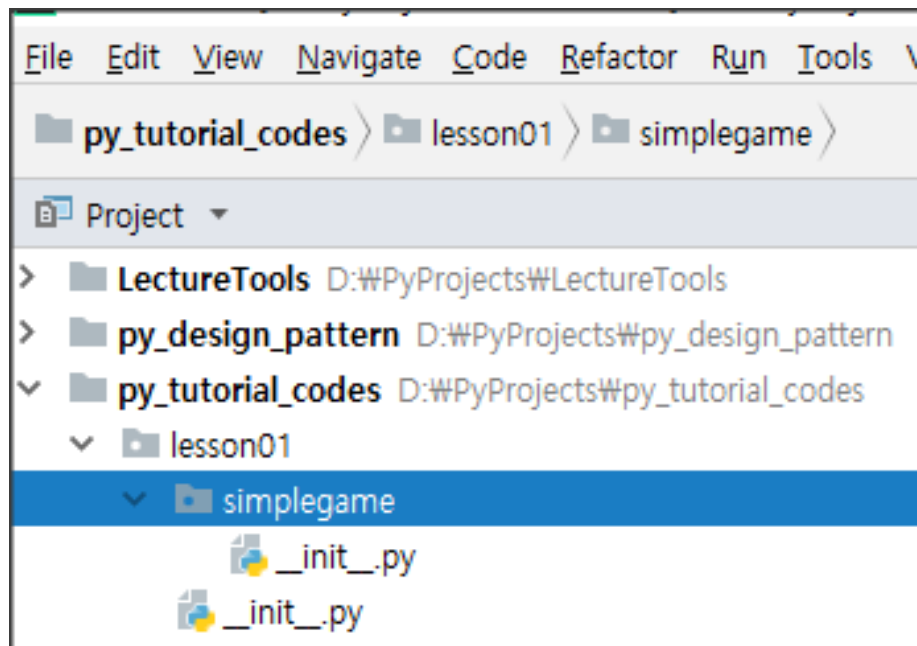


파이썬 패키지 생성 선택

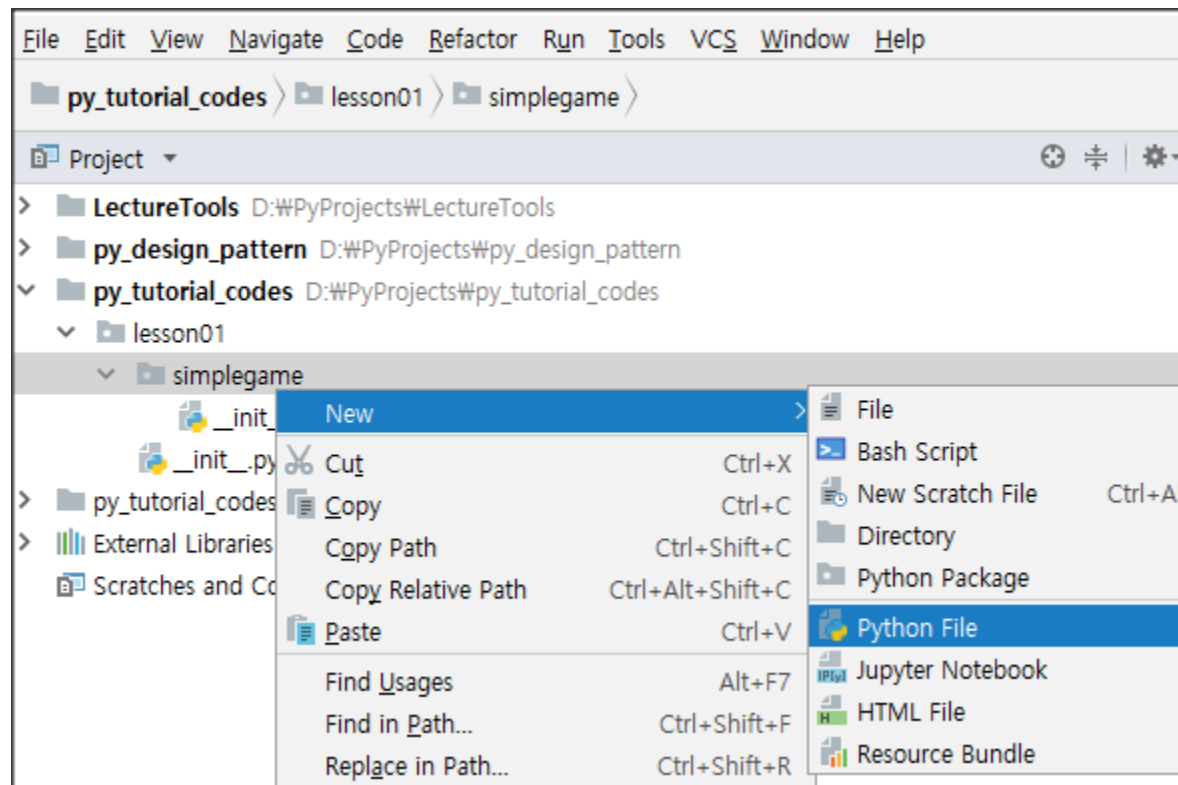


패키지 생성

- lesson01 & lesson01.simplegame 생성

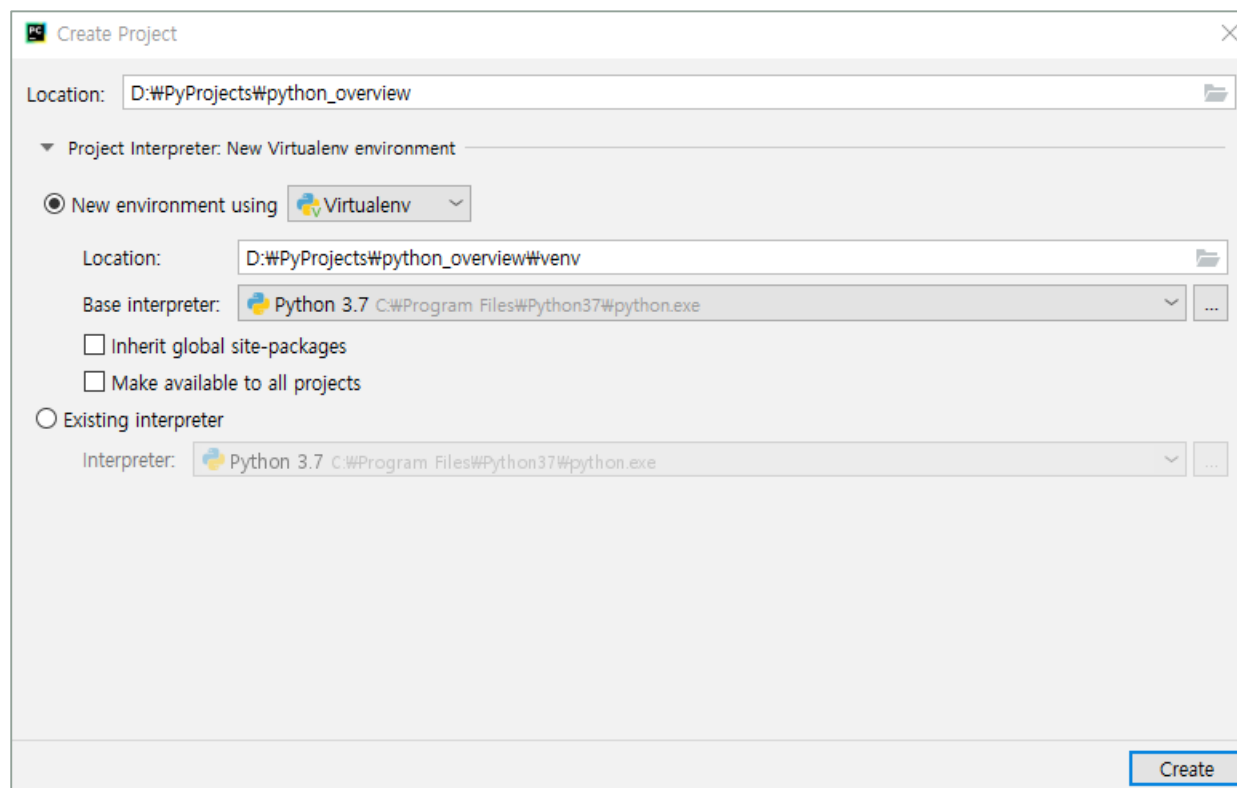


파이썬 파일 추가



프로젝트 생성

- 프로젝트 이름: python_overview
- Virtualenv 사용



실습 01: 프로그램 실행 - 함수 사용

- 파일 이름: hello_using_def.py
- 디렉토리: lesson01

```
#!/usr/bin/env python  # <1>

def say_hello():  # <2>
    print("Hello, world.")  # <3>

if __name__ == "__main__":  # <4>
    say_hello()
```

- <1> #!로 시작하는 첫 번째 줄은 쉬뱅(shebang)이라고 하며, 리눅스와 같은 운영 체제에서 스크립트가 어떻게 실행될지를 나타낸다. 윈도우즈 운영 체제에서는 쉬뱅 줄이 별 의미가 없으며, 확장자 .py에 의한 파일 연관(file association)에 의해 실행된다.
- <2> 함수 정의는 def로 시작한다.
- <3> print() 함수를 사용하여 표준 출력을 한다.
- <4> __name__은 시스템 정의 이름으로, 표준 입력 또는 상호작용 프롬프트로부터 읽혀져 실행될 때 모듈의 __name__은 “__main__”으로 설정된다. \$ python hello_using_def.py → __name__ 은 “__main__”이 됨.

수정 01-1

- 실습 01의 실행 결과가 아래와 같이 되도록 say_hello() 함수를 수정하십시오.
- 수정한 프로그램의 파일 이름: hello_using_def_m1.py

[실행 결과: say_hello("")] → 매개변수 사용
Hello, world.

[실행 결과: say_hello("Tom")] → 매개변수 사용
Hello, Tom.

```
1      #!/usr/bin/env python # <1>
2
3
4      def say_hello(name): # <2>
5          [Redacted]
6
7
8
9
10
11     if __name__ == "__main__": # <4>
12         say_hello("")
13         say_hello("Tom")
```

수정 01-2

- 수정 01의 실행 결과가 아래와 같이 되도록 say_hello() 함수를 수정하시오.
- 수정한 프로그램의 파일 이름: hello_using_def_m2.py

[실행 결과: say_hello()] → 매개변수의 기본 값 사용
Hello, world.

[실행 결과: say_hello("Tom")] → 전달된 인자 값 사용
Hello, Tom.

```
1      #!/usr/bin/env python # <1>
2
3
4      def say_hello(name:                 ): # <2>
5                          
6
7
8      if __name__ == "__main__": # <4>
9          say_hello()
10         say_hello("Tom")
```

실습 02: 프로그램 실행 - 클래스 사용

- 파일 이름: hello_using_class.py

```
#!/usr/bin/env python

class Greeter:
    def __init__(self): # 생성자
        pass

    def say_hello(self):
        print("Hello, world.")

if __name__ == "__main__":
    gildong = Greeter()
    gildong.say_hello()
```

수정 02

- 실습 02의 실행 결과가 아래와 같이 되도록 say_hello() 메소드를 수정하시오.
- 수정한 프로그램의 파일 이름: hello_using_class_m1.py

[실행 결과: gildong.say_hello()]
→ 매개변수의 기본 값 사용
Hello, world.

[실행 결과: gildong.say_hello("Tom")]
→ 전달된 인자 값 사용
Hello, Tom.

```
1  #!/usr/bin/env python
2
3
4  class Greeter:
5      def __init__(self):
6          pass
7
8      def say_hello(self, name):
9          # TODO: Implement say_hello method
10
11
12  if __name__ == "__main__":
13      gildong = Greeter()
14      gildong.say_hello()
15      gildong.say_hello("Tom")
```

줄 구조

- 논리 줄 (logical lines)
 - NEWLINE으로 끝남
- 물리 줄 (physical lines)
 - Unix – ASCII LF
 - Windows – ASCII CR LF
- 주석
 - 해쉬 문자('#')로 시작
- 인코딩
 - 파이썬 스크립트의 첫번째 또는 두번째 줄이 아래와 같은 형식으로 보여질 때
-*- coding: UTF-8 -*-
 - 기본 인코딩
 - Python 2.x: ASCII
 - Python 3.x: UTF-8

- 줄 합치기

- 명시적으로 두 줄 이상의 물리적 줄을 하나의 논리 줄로 합치고자 할 때는 '₩' 문자 사용

```
if 1900 < year < 2100 and 1 <= month <= 12 ₩  
    and 1 <= day <= 31 and 0 <= hour < 24 ₩  
    and 0 <= minute < 60 and 0 <= second < 60:    # Looks like a valid date  
    return 1
```

→ Q. 이상한 점은 없나요?

- 묵시적 줄 합치기

- 괄호 parentheses (), 대괄호 square brackets [], 또는 중괄호 curly braces {} 는 역슬래시 문자 없이 하나 이상의 물리적 줄로 분리 가능

```
month_names = ['Januari', 'Februari', 'Maart',      # These are the  
               'April',   'Mei',      'Juni',      # Dutch names  
               'Juli',    'Augustus', 'September', # for the months  
               'Oktober', 'November', 'December']  # of the year
```

- 들여쓰기 (Indentation)

- 문장 블록을 구분하는 기준
- 공백 문자 또는 탭의 개수로 구분 (이 둘을 섞어 사용하는 것은 좋지 않음)
- 오류 없는 들여쓰기 예

```
def perm(l):  
    # Compute the list of all permutations of l  
    if len(l) <= 1:  
        return [l]  
    r = []  
    for i in range(len(l)):  
        s = l[:i] + l[i+1:]  
        p = perm(s)  
        for x in p:  
            r.append(l[i:i+1] + x)  
    return r
```

- 오류 있는 들여쓰기 예

```
def perm(l):  
for i in range(len(l)):          # error: first line indented  
    s = l[:i] + l[i+1:]          # error: not indented  
    p = perm(l[:i] + l[i+1:])    # error: unexpected indent  
    for x in p:  
        r.append(l[i:i+1] + x)  
    return r                     # error: inconsistent dedent
```

인코딩 선언

- 기본 문법: `# -*- coding: <encoding-name> -*-`
- Python2에서는 한글 사용 시 명시적 선언 필요
 - `# -*- coding: CP949 -*-`
 - `# -*- coding: UTF-8 -*-`
- Python3에서는 'UTF-8'을 기본적으로 사용

식별자 (Identifiers)

- 식별자는 UAX-31과 PEP3131 준수
 - UAX-31: Unicode identifier and pattern syntax
 - PEP 3131: Supporting Non-ASCII Identifiers
- 문자 수는 제한 없으며, 대소문자는 구분
- 식별자 규칙

```
identifier ::= xid_start xid_continue*  
id_start  ::= <all characters in general categories Lu, Ll, Lt, Lm, Lo, Nl, the underscore,  
               and characters with the Other_ID_Start property>  
id_continue ::= <all characters in id_start, plus characters in the categories Mn, Mc, Nd,  
               Pc and others with the Other_ID_Continue property>  
xid_start  ::= <all characters in id_start whose NFKC normalization is in "id_start xid_continue*>  
xid_continue ::= <all characters in id_continue whose NFKC normalization is in "id_continue*>
```

핵심어 (Keywords)

- Python 핵심어 (또는 예약어 reserved words)

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	

특별한 의미가 있는 식별자

- `_`
 - 대화식 인터프리터가 마지막 평가 결과를 저장하는 데 사용
 - 대화식 인터프리터에서 사용되지 않을 때는 의미 없음
 - 참고: <https://stackoverflow.com/questions/6930144/underscore-vs-double-underscore-with-variables-and-methods>
- `--`
 - 시스템 정의 이름
 - 참고 URL: <https://docs.python.org/3/reference/datamodel.html#specialnames>
- `__`
 - 클래스의 비공개 이름(private names)
 - 참고 URL: <https://docs.python.org/3/reference/expressions.html#atom-identifiers>

(참고) `'_'` 두 개가 연속으로 되어 있음을 명시적으로 보이기 위해 `'_'`과 `'_'` 사이에 공백 추가함.

'_' 사용 방법

From [PEP 8](#):

- `_single_leading_underscore` : weak "internal use" indicator. E.g.

```
from M import *
```

does not import objects whose name starts with an underscore.

- `_single_trailing_underscore_` : used by convention to avoid conflicts with Python keyword, e.g.

```
Tkinter.Toplevel(master, class_='ClassName')
```

- `__double_leading_underscore` : when naming a class attribute, invokes name mangling (inside class `FooBar`, `__boo` becomes `_FooBar__boo`; see below).
- `__double_leading_and_trailing_underscore__` : "magic" objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.

리터럴

- 리터럴: 내장 자료형의 상수 값 표현을 위한 표기법
- 종류
 - 문자열 리터럴
 - 바이트 리터럴
 - 형식화된 문자열 리터럴
 - 정수 리터럴
 - 부동소수 리터럴
 - 허수 리터럴

```
literal ::= stringliteral | bytesliteral  
          | integer | floatnumber | imagnumber
```


문자열 리터럴

• 문법

```
stringliteral ::= [stringprefix](shortstring | longstring)
stringprefix  ::= "r" | "u" | "R" | "U" | "f" | "F"
               | "fr" | "Fr" | "fR" | "FR" | "rf" | "rF" | "Rf" | "RF"
shortstring   ::= """ shortstringitem* """ | ''' shortstringitem* '''
longstring    ::= """ longstringitem* """ | ''' longstringitem* '''
shortstringitem ::= shortstringchar | stringescapeseq
longstringitem  ::= longstringchar | stringescapeseq
shortstringchar ::= <any source character except "\" or newline or the quote>
longstringchar  ::= <any source character except "\">
stringescapeseq ::= "\" <any source character>
```

• 사용 예

triple-quotes

```
>>> 'hello'
'hello'
>>> "hello"
'hello'
>>> """ 긴문자열
표시방법
"""
' 긴문자열\n표시방법\n'
>>> """ 긴문자열
표시방법
"""
' 긴문자열\n표시방법\n'
>>> |
```

```
>>> foo = 'hello\nworld\n안녕하세요'
>>> print(repr(foo))
'hello\nworld\n안녕하세요'
>>> print(str(foo))
hello
world
안녕하세요
>>> print(ascii(foo))
'hello\nworld\nWuc548Wub155Wud558Wuc138Wuc694'
```

```
>>> help(repr)
>>> help(str)
>>> help(ascii)
```

바이트 문자열

- 문법

```
bytesliteral ::= bytesprefix(shortbytes | longbytes)
bytesprefix  ::= "b" | "B" | "br" | "Br" | "bR" | "BR" | "rb" | "rB" | "Rb" | "RB"
shortbytes   ::= """ shortbytesitem* """ | ''' shortbytesitem* '''
longbytes    ::= """ longbytesitem* """ | ''' longbytesitem* '''
shortbytesitem ::= shortbyteschar | bytesescapeseq
longbytesitem  ::= longbyteschar | bytesescapeseq
shortbyteschar ::= <any ASCII character except "\" or newline or the quote>
longbyteschar  ::= <any ASCII character except "\">
bytesescapeseq ::= "\" <any ASCII character>
```

- 사용 예

```
>>> b'hello'
b'hello'
>>> b'안녕'
SyntaxError: bytes can only contain ASCII literal characters.
>>>
```

확장 문자(Escape Sequences)

Escape Sequence	Meaning	Notes
<code>\\</code>	Backslash (\)	
<code>\'</code>	Single quote (')	
<code>\"</code>	Double quote (")	
<code>\a</code>	ASCII Bell (BEL)	
<code>\b</code>	ASCII Backspace (BS)	
<code>\f</code>	ASCII Formfeed (FF)	
<code>\n</code>	ASCII Linefeed (LF)	
<code>\r</code>	ASCII Carriage Return (CR)	
<code>\t</code>	ASCII Horizontal Tab (TAB)	
<code>\v</code>	ASCII Vertical Tab (VT)	
<code>\ooo</code>	Character with octal value <i>ooo</i>	(1,3)
<code>\xhh</code>	Character with hex value <i>hh</i>	(2,3)

- 문자열 리터럴에 사용되는 escape sequences

Escape Sequence	Meaning	Notes
\N{name}	Character named <i>name</i> in the Unicode database	(4)
\uxxxx	Character with 16-bit hex value xxxx	(5)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx	(6)

- 참고. 문자열 리터럴 연결(concatenation)

```
>>> "hello" 'world'
'helloworld'
>>> "hello" + "world"
'helloworld'
>>> "hello" + "world"
'helloworld'
```

형식화된 문자열 리터럴 f-string (v3.6)

- 형식화된 문자열 리터럴(formatted string literal or f-string)
 - 'f' 또는 'F'로 시작
 - 대체 필드: 중괄호 {}
 - 실행 시 평가되는 표현식
- 문법

```
f_string      ::= (literal_char | "{" | "}" | replacement_field)*  
replacement_field ::= "{" f_expression ["!" conversion] [":" format_spec] "}"  
f_expression    ::= (conditional_expression | "*" or_expr)  
                  ("," conditional_expression | "," "*" or_expr)* [","]  
                  | yield_expression  
conversion      ::= "s" | "r" | "a"  
format_spec     ::= (literal_char | NULL | replacement_field)*  
literal_char    ::= <any code point except "{", "}" or NULL>
```

- 변환 필드

- !s: str() 함수 호출
- !r: repr() 함수 호출
- !a: ascii() 함수 호출

```
>>> name = 'Fred'
>>> f'His name is {name!r}.'
"His name is 'Fred'."
>>> f'His name is {name}.'
'His name is Fred.'
>>> f'His name is {name!s}.'
'His name is Fred.'
>>> f'His name is {name!a}.'
"His name is 'Fred'."
```

```
>>> name = '철수'
>>> f'이름은 {name!r}입니다.'
"이름은 '철수'입니다."
>>> f'이름은 {name}입니다.'
'이름은 철수입니다.'
>>> f'이름은 {name!s}입니다.'
'이름은 철수입니다.'
>>> f'이름은 {name!a}입니다.'
"이름은 '\\ucca0\\uc218'입니다."
```

정수 리터럴

- 문법

```
integer    ::= decinteger | bininteger | octinteger | hexinteger
decinteger ::= nonzerodigit ([ "_" ] digit)* | "0"+ ([ "_" ] "0")*
bininteger ::= "0" ("b" | "B") ([ "_" ] bindigit)+
octinteger  ::= "0" ("o" | "O") ([ "_" ] octdigit)+
hexinteger  ::= "0" ("x" | "X") ([ "_" ] hexdigit)+
nonzerodigit ::= "1"..."9"
digit       ::= "0"..."9"
bindigit    ::= "0" | "1"
octdigit    ::= "0"..."7"
hexdigit    ::= digit | "a"..."f" | "A"..."F"
```

- '_': 가독성 목적으로 사용되며, 실제 값 계산 시에는 무시 (v3.6)
- 0이 아닌 정수 값에는 숫자 앞에 0이 올 수 없음
- 사용 예

```
7    2147483647                0o177    0b100110111
3    79228162514264337593543950336  0o377    0xdeadbeef
    100_000_000_000                0b_1110_0101
```

부동 소수 리터럴

- 문법

```
floatnumber ::= pointfloat | exponentfloat
pointfloat  ::= [digitpart] fraction | digitpart "."
exponentfloat ::= (digitpart | pointfloat) exponent
digitpart   ::= digit ("_" digit)*
fraction    ::= "." digitpart
exponent    ::= ("e" | "E") ["+" | "-"] digitpart
```

- 정수부와 지수부는 항상 기수 10을 사용하는 것으로 해석

- 사용 예

```
3.14  10.  .001  1e100  3.14e-10  0e0  3.14_15_93
```


허수 리터럴

- 문법

```
imagnumber ::= (floatnumber | digitpart) ("j" | "J")
```

- 사용 예

```
3.14j 10.j 10j .001j 1e100j 3.14e-10j 3.14_15_93j
```

- 참고: 복소수(complex number)
 - (부동 소수 + 허수)로 표시 → 예. (3+4j)

연산자

- 연산자 토큰

+	-	*	**	/	//	%	@
<<	>>	&		^	~		
<	>	<=	>=	==	!=		

//: floor division operator

?: modulo operator

- 몫 & 나머지 연산자

```
>>> 10 // 2
```

```
5
```

```
>>> 10 % 2
```

```
0
```

```
>>> 11 // 2
```

```
5
```

```
>>> 11 % 2
```

```
1
```

- @

- 참고: PEP 465 – A dedicated infix operator for matrix multiplication
- since Python 3.5

구분 문자 (Delimiters)

- 토큰을 구분하는 데 사용되는 구분 문자

```
( ) [ ] { }  
, : . ; @ = ->  
+= -= *= /= //= %= @=  
&= |= ^= >>= <<= **=
```

- 파이썬에서 사용되지 않는 문자

```
$ ? `
```

실습03: int/float 리터럴

- 파일 이름: literal_usage.py

```
1
2 count = 0
3
4
5 def int_literal():
6     print('\n>>> int_literal(): ')
7     n1 = 1_234_567
8     display_info("n1", n1)
9     n1 = 1234567
10    display_info("n1", n1) # 동일한 객체
11    n1 = n1 + 1
12    display_info("n1", n1) # 서로 다른 객체
13
14    # print('n1 = ' + n1)
15    print('n1 = ' + str(n1)) # +: string concatenation
16    print('n1 = ' + format(n1, ',d'))
17    print('n1 = ' + format(n1, '12,d'))
18    print('n1 = ' + format(n1, '012,d'))
19    print('n1 = %d' % n1)
20    print(f'n1 = {n1:}') # 형식화된 문자열 리터럴 v3.6
21    print(f'Wn1 = {n1:15,d}') # 형식화된 문자열 리터럴 v3.6
```

```
>>> int_literal():
(0) n1: id = 1642628952240, type = <class 'int'>, value = 1234567
(1) n1: id = 1642628952240, type = <class 'int'>, value = 1234567
(2) n1: id = 1642626738128, type = <class 'int'>, value = 1234568
n1 = 1234568
n1 = 1,234,568
n1 =  1,234,568
n1 = 0,001,234,568
n1 = 1234568
n1 = 1,234,568
f'n1 =  1,234,568
```

```
23 n2 = int()
24 display_info("n2", n2)
25 n3 = int('0')
26 display_info("n3", n3)
27
28 n3 = 31
29 # 이진수, 팔진수, 십육진수 변환 함수
30 print(n3, bin(n3), oct(n3), hex(n3))
31
32 # 이진수 0b, 팔진수 0o, 십육진수 0x 표기법
33 result = (n3 == 0b1_1111 == 0o37 == 0x1F)
34 print('n3 == 0b1_1111 == 0o37 == 0x1F -->', result)
```

```
(3) n2: id = 140728208905216, type = <class 'int'>, value = 0
(4) n3: id = 140728208905216, type = <class 'int'>, value = 0
31 0b11111 0o37 0x1f
n3 == 0b1_1111 == 0o37 == 0x1F --> True
```

```
37 def float_literal():
38     print('\n>>> float_literal(): ')
39     f1 = 1_234.567
40     display_info("f1", f1)
41
42     print(format(f1, 'f'))
43     print(format(f1, ',.2f'))
44     print('f1 = %10.2f' % f1)
45     print(f'f1 = {f1:,.4f}') # 형식화된 문자열 리터럴 v3.6 f_string
46
47     f2 = float()
48     display_info("f2", f2)
49
50     result = 3.14 == 0.314e1 == 0.0314E2
51     print('3.14 == 0.314e1 == 0.0314E2 -->', result)
```

```
>>> float_literal():
(5) f1: id = 1642597888408, type = <class 'float'>, value = 1234.567
1234.567000
1,234.57
f1 = 1234.57
f1 = 1,234.5670
(6) f2: id = 1642597888552, type = <class 'float'>, value = 0.0
3.14 == 0.314e1 == 0.0314E2 --> True
```

```
54 def bool_literal():  
55     print('\n>>> bool_literal(): ')  
56     b1 = True  
57     display_info("True", b1)  
58  
59     b2 = bool()  
60     display_info("bool()", b2)  
61  
62     b3 = bool('False')  
63     display_info("bool('False')", b3)  
64  
65     b4 = bool('test')  
66     display_info("bool('test')", b4)  
67  
68     b5 = True and False  
69     display_info("True and False", b5)
```

```
>>> bool_literal():  
(7) True: id = 140728208374096, type = <class 'bool'>, value = True  
(8) bool(): id = 140728208374128, type = <class 'bool'>, value = False  
(9) bool('False'): id = 140728208374096, type = <class 'bool'>, value = True  
(10) bool('test'): id = 140728208374096, type = <class 'bool'>, value = True  
(11) True and False: id = 140728208374128, type = <class 'bool'>, value = False
```

```
72 def display_info(name, value):  
73     global count  
74     print("(%s) %s: id = %s, type = %s, value = %s" %  
75         (count, name, id(value), type(value), value))  
76     count += 1  
77  
78  
79 if __name__ == "__main__":  
80     int_literal()  
81     float_literal()  
82     bool_literal()
```