파이썬(Python)의 개요

https://wikidocs.net/book/1

파이썬(Python)은 무엇인가?

사전적 의미

고대 신화 속의 파르나수스 (Parnassus) 산의 동굴에 살던 큰 뱀으로서, 아폴로가 델파이에서 파이썬을 퇴치했다

특징

"인간다운 언어이다."
"문법이 쉬워 빠르게 학습할 수 있다."
"강력하다."
"공짜다."
"프로그래밍이 재미있다."
"개발 속도가 빠르다."

사용 범위

시스템 유틸리티
GUI 프로그래밍
C/C++ 결합
CGI 프로그래밍
수치 연산 프로그래밍
데이터 베이스 프로그래밍
하지만, 성능 위주, 하드웨어 관련 프로그래밍에는 어울리지 않음. 주로 다른 언어로 개발 후 결합시킴.

```
if 4 in [1,2,3,4]: print ("4 존재" )
```

```
languages = ['python', 'perl', 'c', 'java']

for lang in languages:
    if lang in ['python', 'perl']:
        print ("%6s need interpreter" % lang)
    elif lang in ['c', 'java']:
        print ("%6s need compiler" % lang)
    else:
        print ("should not reach here" )
```

자료형(Data Type)

변수는 타입 선언이 필요 없음. (타입 유추)

타입	예제	기타
숫자형 (Number)	정수:123, -345, 0, 실수:123.45, -1234.5, 3.4e10 8진수: 0o34, 0o25, 16진수: 0x2A, 0xFF	
문자열형 (String)	"Hello World", 'Hello World' """ Hello World! My Name… """	A[:], A[0:2], A[1:] A+B, "%d,%d"%(1,2) A[1]='New'=> Error!!
리스트 (List)	[1, 2, 'abc']	문자형과 비슷하지만, 값 수정 가능. A[2]='New'
튜플 (Tuple)	(1, 2, 'abc')	리스트와 비슷하지만 값 수정 불가.
딕셔네리 (Dictionary)	{ 1: 'abc', 3: 'def'}	A[1] A[1]='New'
이진 값 (Boolean)	참 : True, '…', […], 0 이 아닌 수 거짓 : False, '','""", [], (), {}, 0, None	== != > < … A in B A not in B
변수 사용	A=1 A,B=1,2 (A,B)=(1,2) A=B=1 del(A) **A=B A=B[:] A=copy(B)** A is B	

제어(Control Flow) 명령

블록(block) 형성

‘:’
들여쓰기(Indentation)
빈 블록(Empty Block)
pass

조건분기

```
if condition :
  block
elif condition :
  block
else :
  block
```

반복

```
while condition :
  block
for var in list(tuple, string) :
  block
```

ex) for (i,j) in [(1,2),(3,4)] :

```
break
continue
```

```
pocket = ['knife', 'handphone']
watch = 1
if 'money' in pocket :
    print ("택시를 타고 가라 “)
elif watch : pass
else:
    print ("걸어 가라“)
```

```
a = 10
while 1 :
    a = a -1
    if a < 1:
        print ("exit loop.")
        break
while a<10 :
    a=a+1
    if a%2 == 0 : continue
    print(a)
```

```
marks = [90, 25, 67, 45, 80]
for number in range(0, 5) :
    if marks[number] < 60: continue
    print(number)

for i in range(2,10) :
    for j in range(1, 10) :
        print(i*j)
    print('\n‘)
```

함수(Function)

함수 정의

```
def func_name( arguments ) :
  block
return value
return
```

인수(arguments)

```
arg_name1, arg_name2, …
arg_name1, *arg_name2
arg_name1, arg_name2=1
```

람다(lambda) 함수 정의

```
lambda var_list : expression
var = lambda …
```

함수 실행(apply)

```
func_name( arguments )
```

함수 실행 결과 저장

```
var = func_name(…)
```

```
sum(3, 4)
sumprint(1, 2, 3, 5)
say_myself("홍길동", 10, 1)
countdown(5)
a = sum_l( 3, 4 )
```

```
def sum( a, b ) :
    return a + b
def sum_print( a, b ) :
    print("%d+%d=%d" % (a, b, a+b))
def sum_many( *args ) :
    sum = 0
    for i in args:
        sum = sum + i
    return sum
def say_myself( name, old, sex=1 ) :
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if sex : print("남자입니다.")
    else: print("여자입니다." )
def countdown( n ) :
    print(n)
    if n == 0:
        print("0")
    else:
        countdown( n-1 )
sum_l = lambda a, b : a+b
```

```
myList = [lambda a,b:a+b, lambda a,b:a*b]
myList[0](3,4)
myList[1](3,4)
```

클래스(Class)

클래스 정의

```
class cls_name :
  block
class cls_name( parent_cls_names ) :
  block
```

객체 생성

```
obj_name = cls_name( arguments )
```

클래스 변수

정의

```
var = init_value
self.var = value
```

설정

```
obj_name.var = value
self.var = value
```

멤버 함수

```
def func_name( self, arguments ) :
  block
```

클래스 블록 내에 정의

반드시 self(객체 자신)를 포함해야 함.

함수 실행 : self 는 생략해도 됨

```
obj_name.func_name( arguments )
__init__( self, arguments )
__del__( self )
__add__( self, other )
...
```

```
class HousePark :
    firstname = "박"
    def __init__( self, name ) :
        self.fullname = self.firstname + name
    def travel( self,  where ) :
        print("%s, %s여행." % (self.fullname, where))
    def love( self, other ) :
        print("%s, %s 사랑" % (self.fullname, other.fullname) )
    def __add__( self, other ) :
        print("%s, %s 결혼" % (self.fullname, other.fullname) )
    def __del__( self ) :
        print("%s 죽네" % self.fullname)

class HouseKim( HousePark ):
    firstname = "김"
    def travel( self, where, day ) :
        print("%s, %s여행 %d일" % (self.fullname, where, day) )

pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.love(julliet)
pey + julliet
```

모듈(Module)

모듈 로드

```
import mod_name
from mod_name import mod_func_name,...
        from mod_name import *
reload( mode_name )
```

모듈 참조

```
mod_name.var
mod_name.mod_func_name
```

메인 모듈 여부 판단

```
if __name__=="__main__" :
  block
```

모듈 경로 설정

```
import sys
sys.path.append( module_path )
```

```
# mod2.py
PI = 3.141592
class Math:
    def solv(self, r):
        return PI * (r ** 2)
    def sum(a, b):
        return a+b
if __name__ == "__main__":
    print(PI a = Math())
    print(a.solv(2))
    print(sum(PI , 4.4))
```

```
# modtest.py
import mod2
result = mod2.sum(3, 4)
print(result)
```

```
# config.py
a='doo'
```

```
# reload_test.py
import config
import imp
f = open("config.py", 'w')
f.write("a = 'foo'")
f.close()
imp.reload(config)
c = config.a
print(c)
```

예외 처리(Exception)

예외처리

```
try:
  block
except :
  block

try:
  block
except error :
  block

try:
  block
except error as var :
  block
```

```
try:
    4 / 0
except ZeroDivisionError as e :
    print(e)

try:
    f = open("나없는파일.txt", 'r')
except IOError :
    print("쓰기모드로 파일을 엽니다.")
    f = open("나없는파일.txt", 'w')
```

내장함수

함수	설명
abs	절대값
apply	함수 적용
chr	아스키 코드값을 받아 문자출력
cmp	객체 비교
dir	객체의 클래스 변수/함수 나열
divmod	몫과 나머지, (div,mod)
eval	문자열을 프로그램으로 인식 계산
execfile	파이썬 파일 실행
filter	리스트 필터링. filter(func, list)
hex	16진수를 표시하는 문자열로 변환
id	객체 아이디
input	사용자 입력 , input(prompt)
int	정수로 변환
isinstance	클래스의 객체 여부 판단
lambda	람다 함수
len	리스트 길이
list	리스트로 변환

함수	설명
long	정수로 변환
map	리스트에 함수 적용 리턴
max	최대값
min	최소값
oct	8진수 문자열로 변환
open	파일 열기
ord	아스키값 리턴
pow	제곱승
range	해당 범위의 리스트를 생성
raw_input	사용자 입력
reduce	리스트의 모든 값을 함수에 적용
reload	모듈 리로드
repr	객체를 문자열로 변환
str	객체를 문자열로 변환,eval 사용x
tuple	튜플로 변환
type	타입을 리턴

파이썬 구조 분석

Written by 박 철(e2g1234@naver.com)

파이썬과 자바 실행과정

파이썬 실행과정

- ✓소스코드
- ✓어휘분석
- ✓구문분석
- ✓바이트 코드 생성
- ✓가상머신 실행

자바 실행과정

- ✓소스코드
- ✓어휘분석
- ✓구문분석
- ✓바이트 코드 생성
- ✓가상머신 실행

기존 **interpreter**의 단점을 **Python virtual machine (PVM)**의 도입 등을 통해 커버
Python 고유의 **dynamic language**의 편리성
- 전문 프로그래머가 아닌 수학자나 연구자들이 다양한 라이브러리를 쉽게 개발하여 배포할 수 있게 해 주었음
- 이는 강력한 **third party libraries**의 지원으로 이어졌음

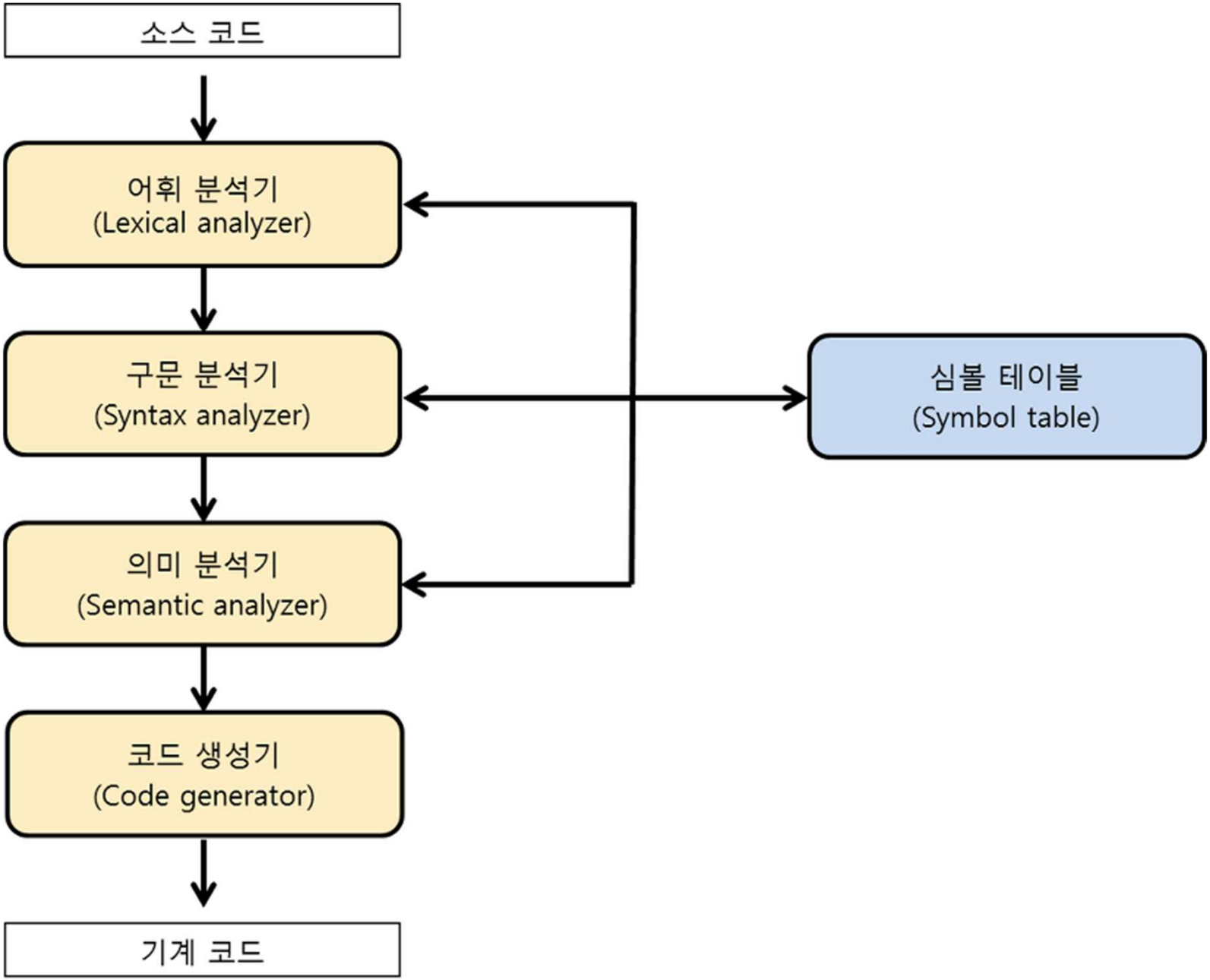
파이썬과 자바의 차이점

- 파이썬 인터프리터, 자바 컴파일러
- 요즘 인터프리터는 컴파일러와 실행과정이 비슷합니다.
- 컴파일러와 차이점
  - 실행시점
    - 인터프리터 : 소스코드 컴파일 중(=해석 중) 실행
    - 컴파일러 : 컴파일이 타 끝난 후 실행
  - 타입시스템

동적 타입 – 변수선언을 하지 않는다.

    - 인터프리터는 동적 타입 : 실행 중에 타입 구성
    - 컴파일러는 정적 타입 : 실행 전에 타입 구성 완료
    - 동적 타이핑을 지원하는 컴파일러도 존재
      - C#, LISP, Objective-C, Julia

컴파일러의 구조


소스 코드
어휘 분석기
(Lexical analyzer)
구문 분석기
(Syntax analyzer)
의미 분석기
(Semantic analyzer)
코드 생성기
(Code generator)
기계 코드
심볼 테이블
(Symbol table)

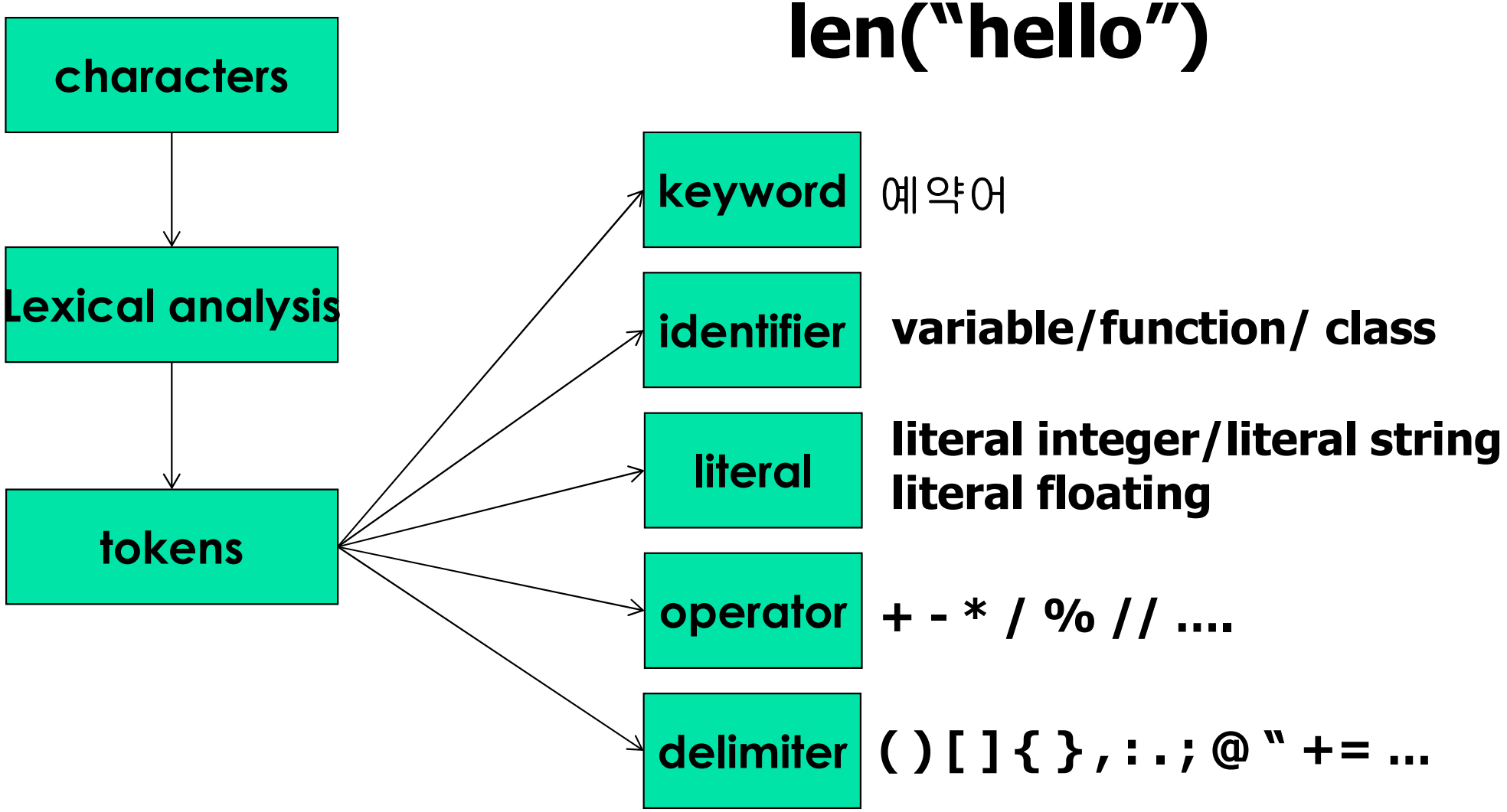
어휘 분서기, 구문 분석기

- 어휘분석기(Lexical analysis)
  - 키워드 정의와 단어 생성 규칙을 정의하고 확인
  - 결과 : 토큰 하나의 토큰은 <토큰이름, 속성값> 토큰(token)의 집합으로 변환
- 구문 분석기(=파서)
  - 토큰 조합 규칙(문법)을 정의하고 확인
  - 결과 : 파스 트리, 구문 트리(AST – Abstract Syntax Tree, 추상 구문 트리)
- 파이썬 파서는 어휘분석기를 내장

파이썬 심볼 테이블

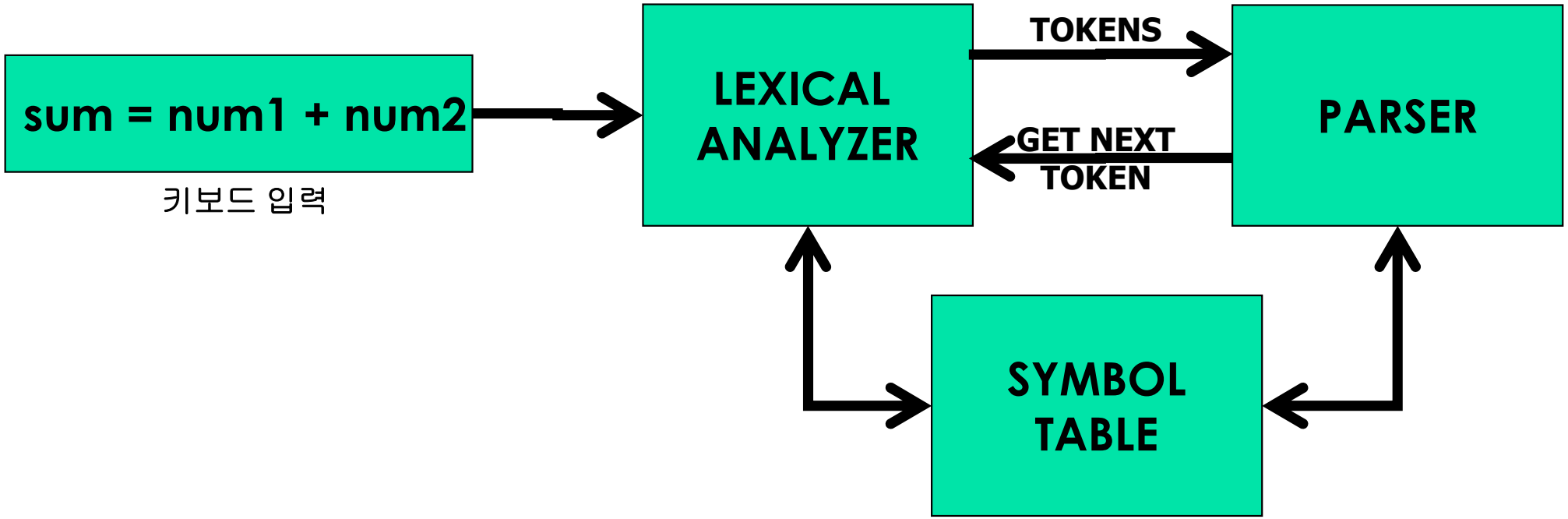
- 식별자(변수, 함수명)의 정보(이름, 타입, 스코프) 를 관리하는 곳
- 보통 딕션나리로 되어 있으며 식별자 이름과 번호를 키, 값으로 저장 파이썬은 locals(), globals()를 통해 심볼테이블 내용을 확인 symtable 모듈로 심볼 테이블을 좀더 심오하게 볼 수 있다.

Lexical analysis



Lexical analysis

len("hello")



sum	**=**	**num1**	**+**	**num2**
identifier	**operator**	**identifier**	**operator**	**identifier**
<id, 1>	<assign, NULL>	<id, 2>	<plus, NULL>	<id, 3>

Lexical analysis & AST

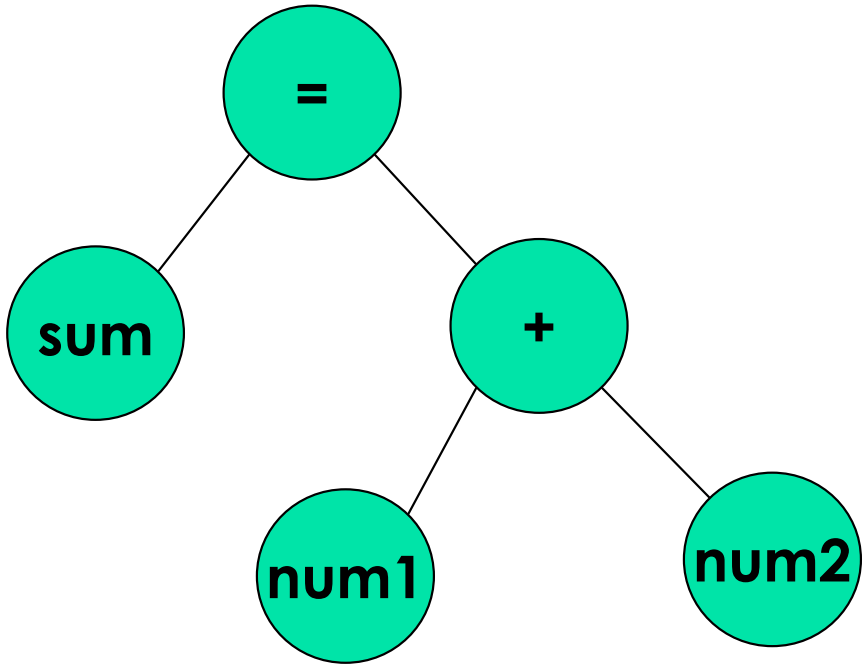
파서 트리 :

구문구조를 토큰들을 단말 노드로 하는 트리형태로 표현한 것

Lexical analysis

sum = num1 + num2

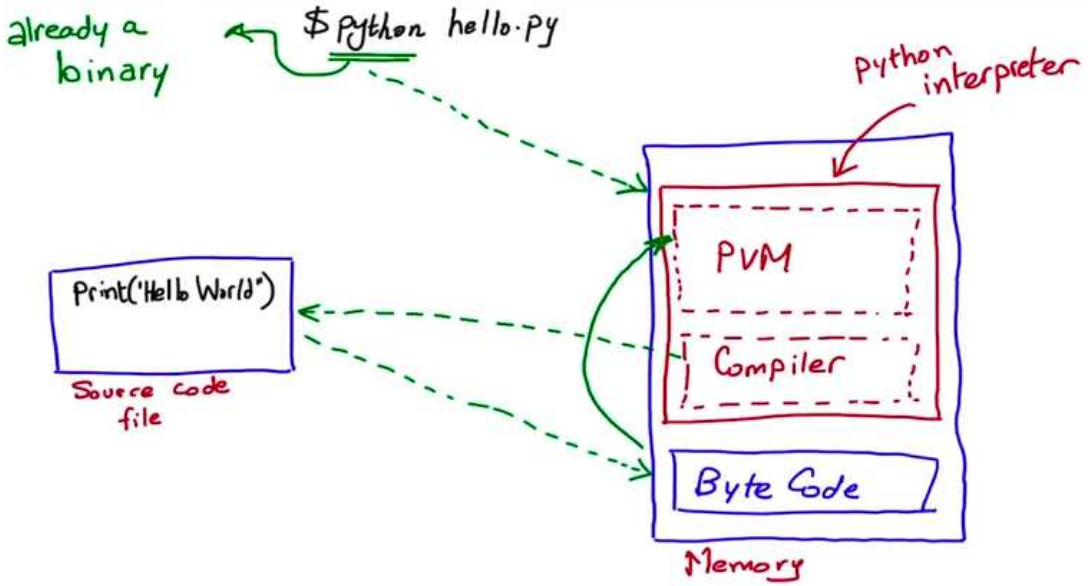
Abstract Syntax Tree



Interpreter

- ✓ Virtual stack machine
- ✓ Excute the bytecode

execution = interpreter(bytecodes)
or
giver life to your code

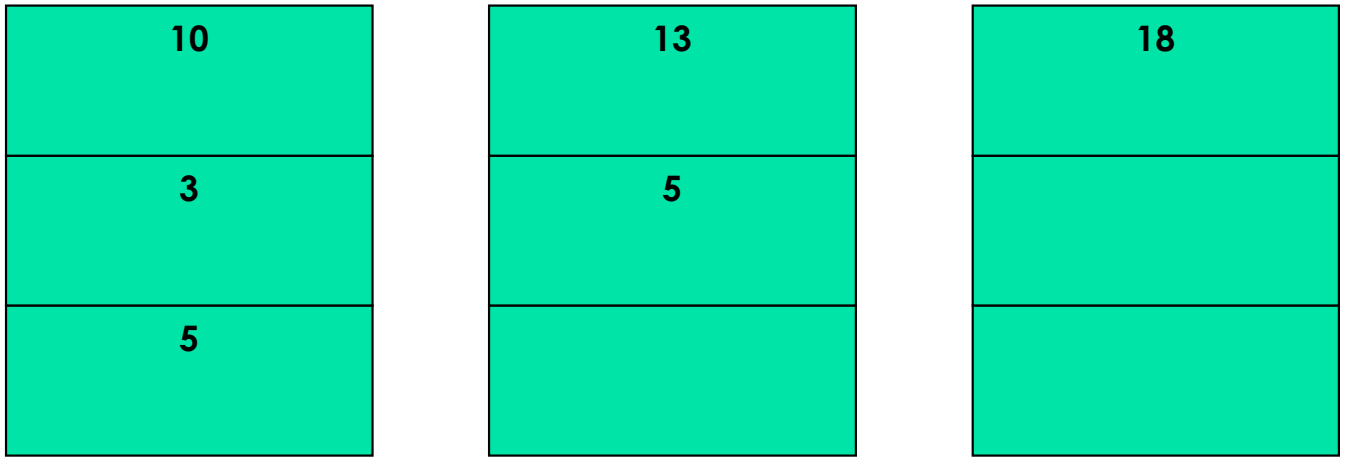


Stack 기반 virtal machine과 register 기반 virtal machine 으로 나눔
Interpeter는 compile와 VM으로 구성

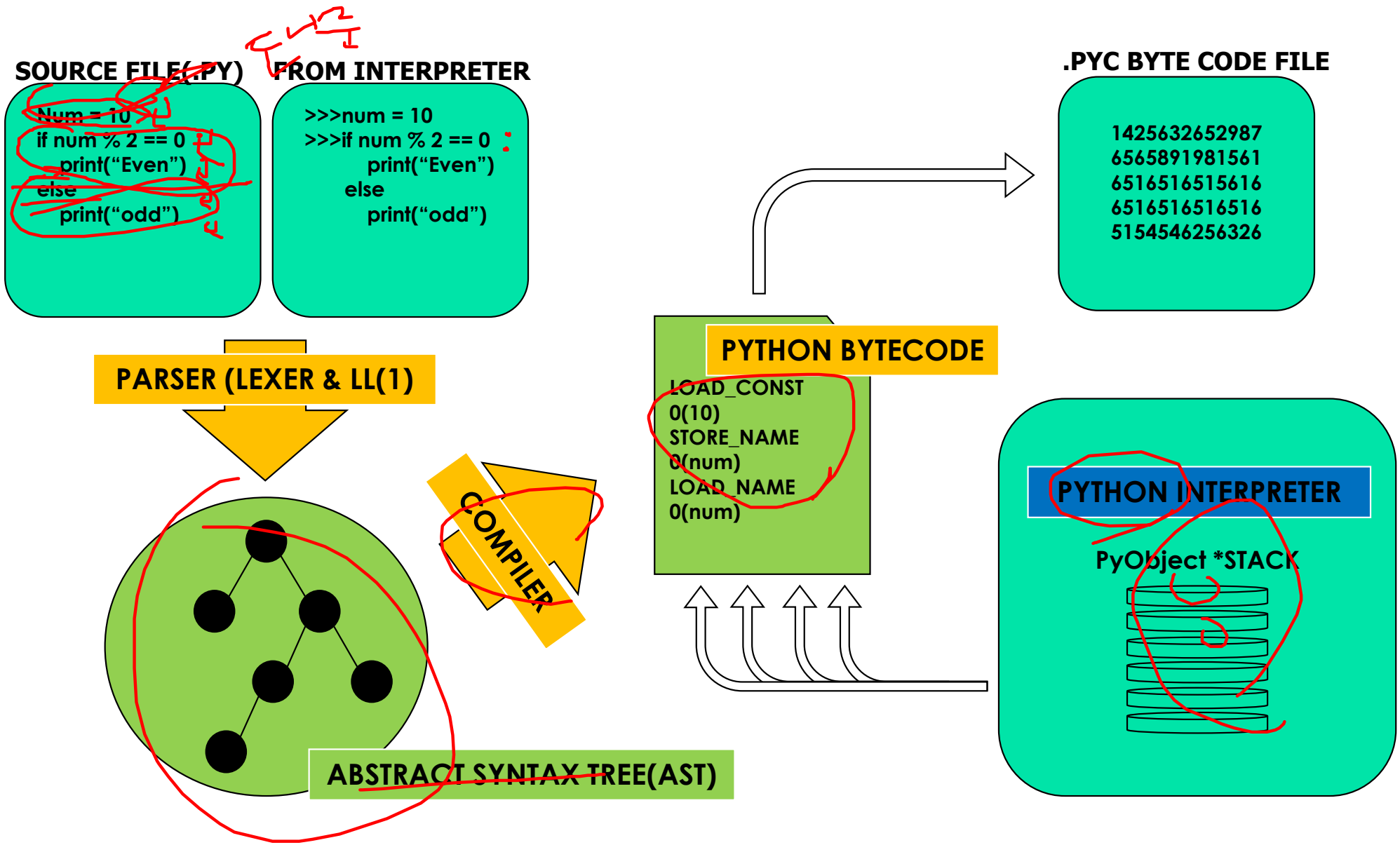
Interpreter

5 + (3 + 10)

PUSH 5
PUSH 3
PUSH 10
ADD
ADD
POP

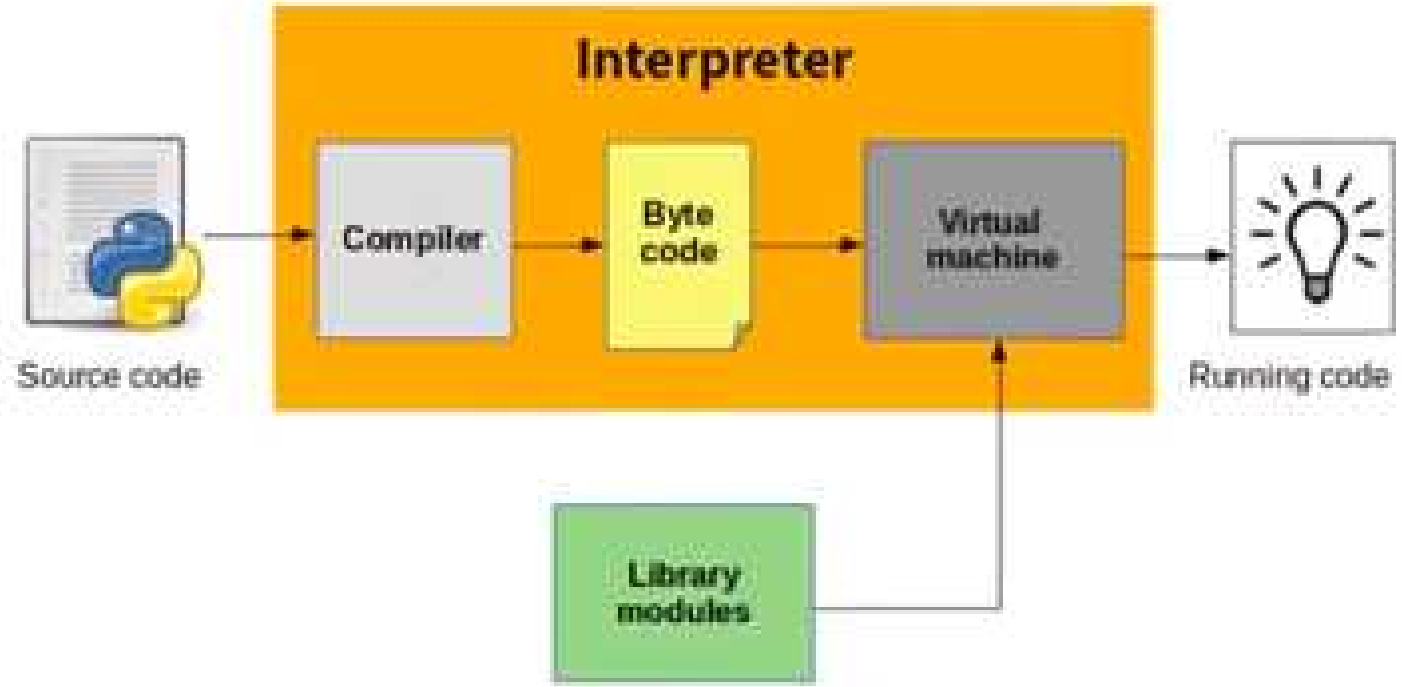


실행과정

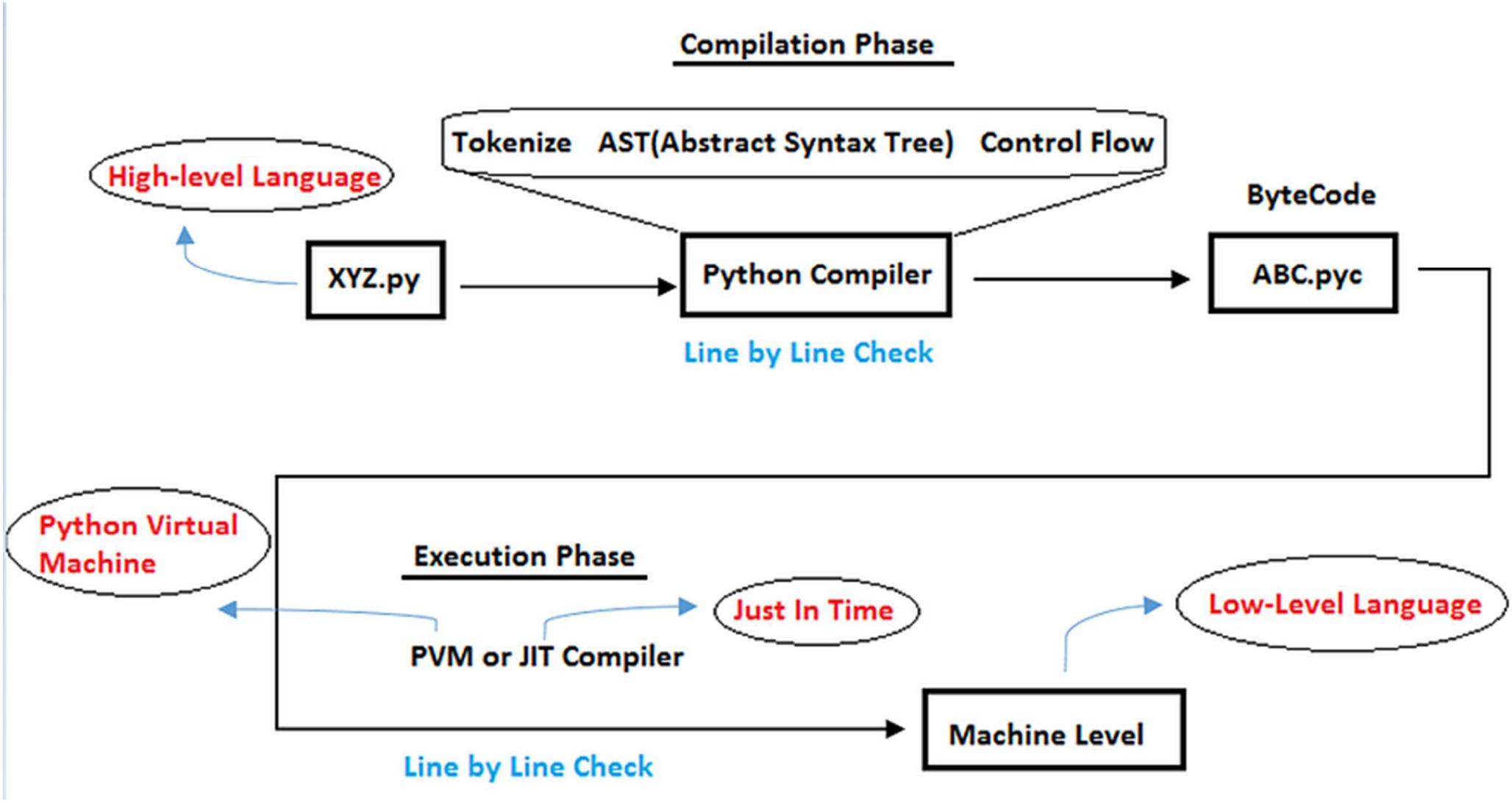

SOURCE FILE(.PY)
Num = 10
if num % 2 == 0
print("Even")
else
print("odd")
FROM INTERPRETER
>>>num = 10
>>>if num % 2 == 0
print("Even")
else
print("odd")
.PYC BYTE CODE FILE
1425632652987
6565891981561
6516516515616
6516516516516
5154546256326
PARSER (LEXER & LL(1)
COMPILER
PYTHON BYTECODE
LOAD_CONST
0(10)
STORE_NAME
0(num)
LOAD_NAME
0(num)
PYTHON INTERPRETER
PyObject *STACK
ABSTRACT SYNTAX TREE(AST)

Inside interpreter

Inside interpreter

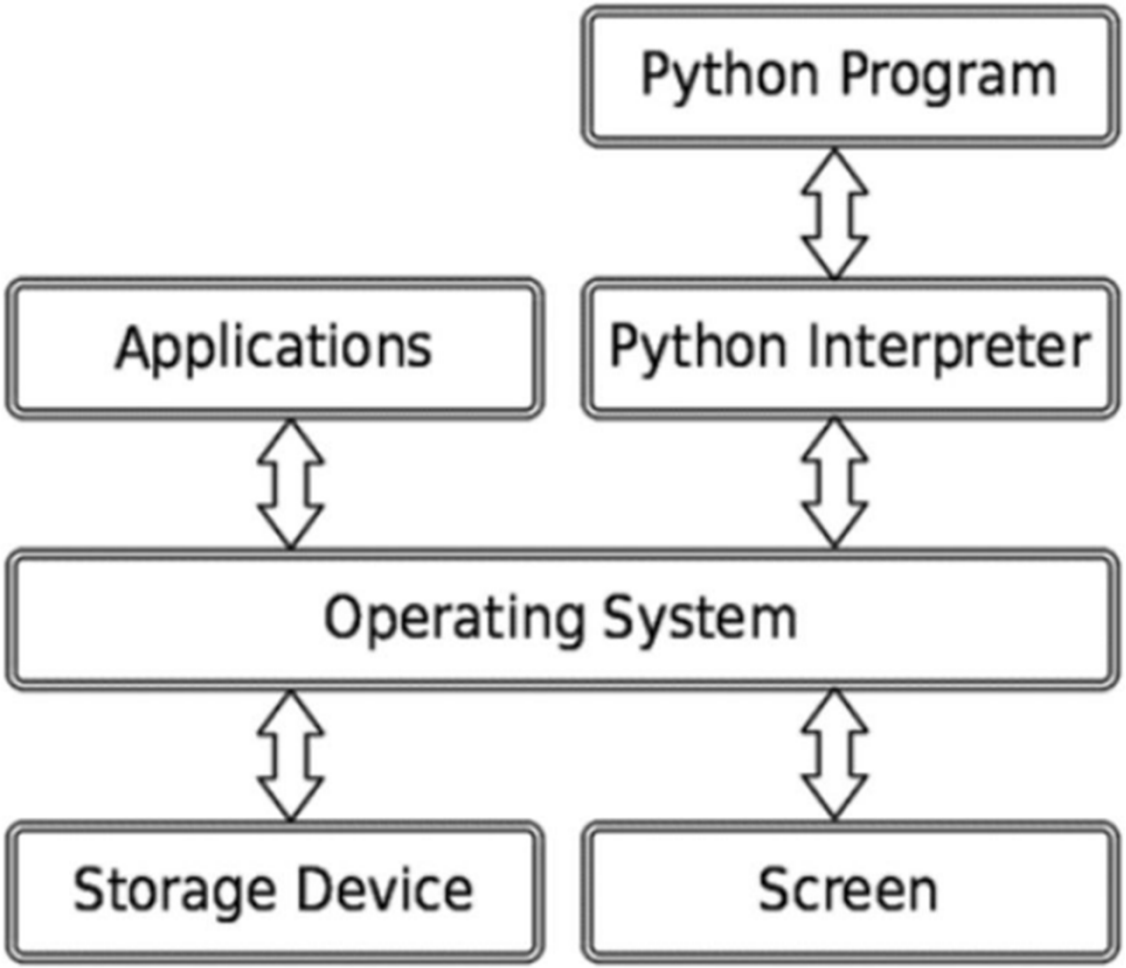


Compilation Phase

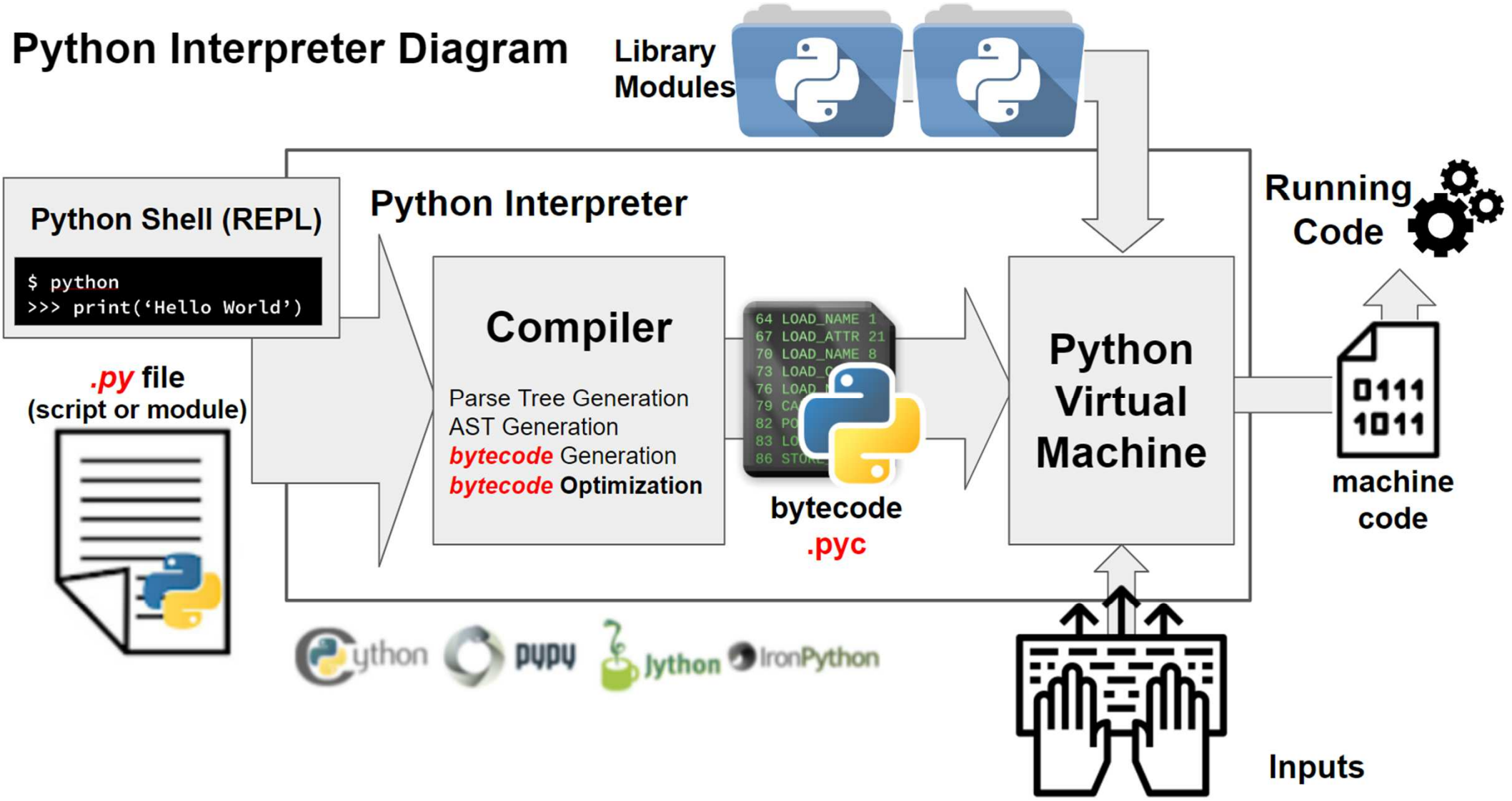


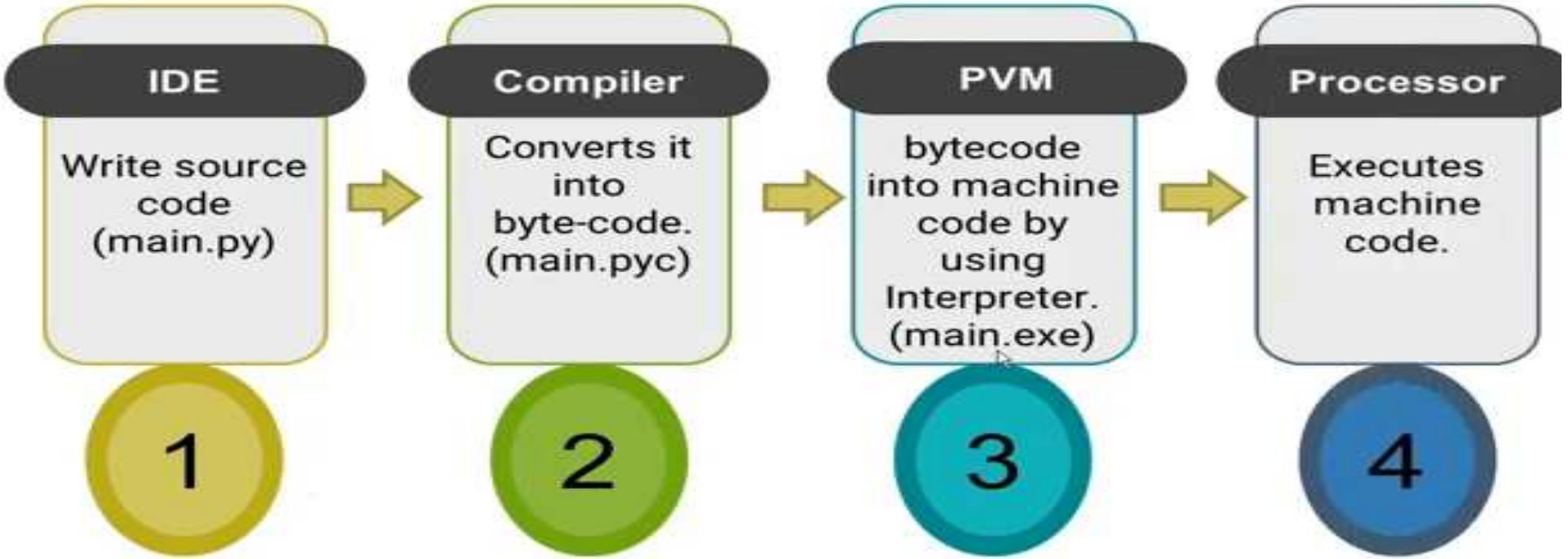
Python Interpreter

Interpreter는 개발자가 작성한 **script**를 **OS**가 알아들을 수 있는 언어로 바꾸어주는 역할을 하기 때문에 통역사라는 이름이 붙음.



Python Interpreter Diagram




IDE
Write source code (main.py)
1
Compiler
Converts it into byte-code. (main.pyc)
2
PVM
bytecode into machine code by using Interpreter. (main.exe)
3
Processor
Executes machine code.
4

파이썬 구조 분석2

Written by 박 철(e2g1234@naver.com)

Key word

키워드 : 예약된 identification - 이름

35개의 키워드를 가지고 있다. 미리 정의된 이름

다음 식별자들은 예약어, 또는 언어의 키워드, 로 사용되고, 일반적인 식별자로 사용될 수 없습니다. 여기 쓰여 있는 것과 정확히 같게 사용되어야 합니다:

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
And	continue	for	lambda	try
As	def	from	nonlocal	while
Assert	del	global	not	with
Async	elif	if	or	yield

Key word

False True None	-> bool **관련**
if else elif / while / for	-> **조건문**
pass break continue	
def return lambda	-> **함수**
class	-> class
import from	-> module
is	-> **존재 연산자**
in	-> **멤버 연산자**
and or not	-> **논리 연산자**
global nonlocal	
del	
async await	-> async - thread **관련**
yield	
try except raise assert finally	-> exception
with as	-> with **구문과** context
manager	

문장의 끝에 ':'이 항상 들어감

if, elif, else, while, for, def, class, try, except, finally, with.

구문이라 함은, 어떤 일를 하도록 하는 지시를 담고 있는 코드 행을 말하는 것
구문에 사용되는 키워드
구문 키워드

구문 열쇠말

if-elif-else	for
while	continue
break	try-except-finally
assert	def
print	del
raise	import

연산자(operator)

연산자(operator) <-> 피연산자(operand : 변수, 상수)

다음과 같은 토큰들은 연산자입니다 :

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

https://docs.python.org/ko/3/reference/lexical_analysis.html

연산자

수학(산술) 연산자

연산자	설명
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
//	나눗셈의 몫
%	모듈로 (나눗셈의 나머지)
**	지수 연산자
+var	단항 덧셈
-var	단항 뺄셈

비교 (관계) 연산자

연산자	설명
>	큼
<	작음
>=	크거나 같음
<=	작거나 같음
!=	같지 않음
==	같음

비트 연산자

연산자	설명
&	논리곱 연산자로서 비트가 두 항에 모두 나타나는 경우 비트를 결과에 복사함
|	논리합 연산자로서 비트가 두 항 중 어느 곳에 나타나는 경우 비트를 결과에 복사함
^	배타적 논리합 연산자로서 어느 한 쪽의 항에만 비트가 존재할 경우 비트를 결과에 복사함
~	부정 연산자로서 비트를 뒤집어서 각 비트에 대하여 정확히 반대를 반환함

복합 대입 연산자

연산자	동치
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b
a //= b	a = a // b
a **= b	a = a** b
a &= b	a = a & b
a |= b	a = a | b
a ^= b	a = a ^ b
a >>= b	a = a >> b
a <<= b	a = a << b

논리 연산자

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)

이동 연산자(비트연사자에 포함됨)

x<<n	왼쪽으로 이동 (숫자 x에 2를 n번 곱한 것과 동등함)
x>>n	오른쪽으로 이동 (숫자 x를 2로 n번 나눈 것과 동등함)

멤버 연산자 : in

존재 연산자 : is

연산자 우선순위

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

delimiter

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

- 마침표는 실수와 허수 리터럴에서도 등장할 수 있습니다.

- 연속된 마침표 세 개는 생략부호 리터럴(ellipsis literal)이라는 특별한 의미가 있습니다.

- 목록 후반의 증분 대입 연산자(augmented assignment operator)들은 어휘적으로는 구분자로 기능하지만, 동시에 연산을 수행합니다.

- 다음의 인쇄되는 ASCII 문자들은 다른 토큰들 일부로서 특별한 의미가 있거나, 그렇지 않으면 어휘 분석기에 유의미합니다:

이스케이프 코드

- 문자열 예제에서 여러 줄의 문장을 처리할 때 백슬래시 문자와 소문자 n을 조합한 \n 이스케이프 코드를 사용했다.
- 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"이다.
- 주로 출력물을 보기 좋게 정렬하는 용도로 이용된다.
- 몇 가지 이스케이프 코드를 정리하면 다음과 같다.
- 이중에서 활용빈도가 높은 것은 \n, \t, \\, \', \"이다.
- 나머지는 프로그램에서 잘 사용되지 않는다.

코드	설명
\n	개행 (줄바꿈)
\t	수평 탭
\\	문자 "\"
\´	단일 인용부호(´)
\"	이중 인용부호(")
\r	캐리지 리턴
\f	폼 피드
\a	벨 소리
\b	백 스페이스
\000	널문자

문자열 포맷

문자열 포맷 코드

- 문자열 포매팅 예제에서는 대입시켜 넣는 자료형으로 정수와 문자열을 사용했으나 이 외에도 다양한 것들을 대입시킬 수 있다.
- 문자열 포맷 코드로는 다음과 같은 것들이 있다.

코드	설명
%s	문자열 (String)
%c	문자 1개(character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)