

# CHAPTER 03

# OPENCV와 파이썬

PART 01 영상 처리 개요 및 OPENCV 소개

# CONTENTS

3.1 파일/자료 구조

3.2 물리적/논리적 명령행

3.3 연산자

3.4. 기본 명령문

3.5 함수와 라이브러리

3.6 넘파이(NUMPY) 패키지

## 3.1 파이썬 자료 구조

- ◆ 3.1.1 상수(constant)와 리터럴(literal)
- ◆ 3.1.2 변수(variable)
- ◆ 3.1.3 자료 구조 - 리스트, 튜플, 사전(dictionary), 집합(set)

### 3.1.1 상수(constant)와 리터럴(literal)

- ◆ 리터럴 - 값 자체
- ◆ 상수 - 항상 똑같은 값이 저장된 곳
- ◆ 파이썬에서는 C언어의 const 나 #define과 같은 키워드 제공 않음
  - ❖ 원칙적으로 상수 사용 안됨
  - ❖ 대문자로 만든 변수를 상수로서 사용

〈표 3.1〉 리터럴의 종류

	종류	예
숫자 리터럴	정수, 실수, 복소수	4, 3.141592, 3+5j
문자 리터럴	따옴표로 묶인 문자	"This is Python", '작은 따옴표도 됨'
논리값 리터럴	True, False	True, False
특수 리터럴	None	None
컬렉션 리터럴	리스트, 튜플, 집합(set), 사전(dictionary)	[1, 2, 3, 4], (3, 5, 7), {'a' , 'b' } {'a': 'apple', 'b': 'ball'}

## 3.1.2 변수(variable)

### ◆ 변수

- ❖ ‘변하는 수’ ?
- ❖ 컴퓨터에서 변하는 것은 기억장치의 어떤 공간에 들어 있는 값
  
- ❖ 변수의 자료형
  - 변수에 값이 처음 할당될 때 그 값의 종류에 의해 결정
  - Numbers
    - ✓ int, long ,float, complex
  - string
  - bool
  
- 열거형(Sequence)
  - ✓ 리스트, 튜플, 사전, 집합

## 3.1.2 변수(variable)

예제 3.1.1

변수의 자료형 – 01.variable.py

### ◆ 예제

```
01 variable1 = 100  
02 variable2 = 3.14  
03 variable3 = -200  
04 variable4 = 1.2 + 3.4j  
05 variable5 = 'This is Python'
```

```
06
```

```
07 variable6 = True
```

# bool 변수 선언

```
08 variable7 = float(variable1)
```

# 자료형 변경

```
09 variable8 = int(variable2)
```

# 자료형 변경

```
10
```

```
11 print('variable1 =' , variable1, type(variable1))
```

# 변수의 값과 자료형 출력

```
12 print('variable2 =' , variable2, type(variable2))
```

```
13 print('variable3 =' , variable3, type(variable3))
```

```
14 print('variable4 =' , variable4, type(variable4))
```

```
15 print('variable5 =' , variable5, type(variable5))
```

```
16 print('variable6 =' , variable6, type(variable6))
```

```
17 print('variable7 =' , variable7, type(variable7))
```

# 실수로 변경되어 소수점 표시됨

```
18 print('variable8 =' , variable8, type(variable8))
```

# 정수로 변경되어 소수점이하 소실

Run: 01.variable

C:\Python\python.exe D:/source/chap03/01.variable.py

variable1 = 100 <class 'int'>

variable2 = 3.14 <class 'float'>

variable3 = -200 <class 'int'>

variable4 = (1.2+3.4j) <class 'complex'>

variable5 = This is Python <class 'str'>

variable6 = True <class 'bool'>

variable7 = 100.0 <class 'float'> → 실수로 변경

variable8 = 3 <class 'int'> → 정수로 변경, 소수 부분 소실

# 문자열 변수 선언

### 3.1.3 열거형(sequence) 객체

- ◆ 파이썬에서는 원소들을 모아 담을 수 있는 다양한 자료 구조 제공
  - ❖ 리스트, 튜플, 사전(dictionary), 집합(set)
- ◆ 리스트
  - ❖ 대괄호([, ])를 둘러싸서 선언
  - ❖ 항목의 추가, 삭제, 값의 변경 가능
- ◆ 튜플
  - ❖ 소괄호((, )) 이용해서 선언
  - ❖ 항목에 대한 변경 불가
- ◆ 사전(dictionary)
  - ❖ 중괄호({, })를 사용해 선언
  - ❖ 객체마다 키를 같이 구성해야 함, 이 키를 이용해서 원소에 쉽게 접근
- ◆ 집합(set)
  - ❖ 원소를 모아 담는 것은 같음
  - ❖ 중복되는 원소들을 하나만 저장해서 원소들의 중복을 없앰

### 3.1.3 열거형(sequence) 객체

예제 3.1.2

자료 구조 - 02.list.py

#### ◆ 예제

```
01 list1 = [1, 2, 3, 4]
02 list2 = [1, 1.5, 'a', 'a', '문자열']
03 tuple1 = (1, 2)
04 tuple2 = (1, 1.5, 'b', 'b', '문자열' )
05 dict1 = { 'name': '배종욱', 'email': 'daum.net'}
06 set1, set2 = set(list2), set(tuple2)

07
08 list1[0] = 5                                # 0번 원소 값 변경
09 list2.insert(3, 'b')                          # 원소 삽입
10 # tuple1[0] = 5                            # 에러 발생- 튜플은 원소 변경 불가
11 dict1['email'] = 'naver.com'                 # 키값으로 접근

12
13 print('list1', list1, type(list1))          # 객체 원소 및 자료형 출력
14 print('list2', list2, type(list2))
15 print('tuple1', tuple1, type(tuple1))
16 print('dict1', dict1, type(dict1))
17 print('set1', set1, type(set1))
18 print('set2', set2, type(set2))
19 print('intersection', set1 & set2 )          # 교집합 구함
```

Run: 02.list

```
C:\Python\python.exe D:/source/chap03/02.list.py
list1 [5, 2, 3, 4] <class 'list'>
list2 [1, 1.5, 'a', 'b', 'a', '문자열'] <class 'list'>
tuple1 (1, 2) <class 'tuple'>
dict1 {'name': '배종욱', 'email': 'naver.com'} <class 'dict'>
set1 {1, '문자열', 'a', 1.5} <class 'set'>
set2 {1, '문자열', 'b', 1.5} <class 'set'>
intersection {1, '문자열', 1.5}
```

## 3.2 물리적/논리적 명령행

◆ 물리적 명령행 - 소스 코드에서 눈으로 보이는 한 행

◆ 논리적 명령행 - 파이썬 인터프리터 관점

예제 3.2.1

명령행 - 03.command\_line.py

```
01 title = '서기 1년 1월 1일부터' \
02     '오늘까지' \
03     '일수 구하기'                                # 3행에 걸쳐 작성 → 1개 논리적 명령행
04 months = [31, 28, 31, 30, 31, 30,
05         31, 31, 30, 31, 30, 31]
06 year, month = 2020, 1                          # 여러개 변수 한행에 선언
07 day = 7; ratio = 365.2425                      # 2개 논리적 명령행
08
09 days = (year -1) * ratio + \
10     sum(months[:month-1]) + day                # 1개 논리적 명령행
11
12 print(title), print(' - 년:', year ), print(' - 월:', month)
13 print(' - 일:', day); print(' * 일수 총합:', int(days))
```

Run: 03.command\_line

```
C:\Python\python.exe D:/source/chap03/03.command_line.py
서기 1년 1월 1일부터 오늘까지 일수 구하기
- 년: 2020
- 월: 1
- 일: 7
* 일수 총합: 737431
```

# 3행에 걸쳐 작성 → 1개 논리적 명령행

# 여러개 변수 한행에 선언

# 2개 논리적 명령행

# 1개 논리적 명령행

### 3.3 연산자

#### ◆ 기본 연산자 및 우선순위

- ❖ 식 (expression) : 결과 또는 값을 만들어내는 코드 조각

식 = 피연산자 (연산자) 피연산자

식 = (연산자) 피연산자

- 연산자(operator): 연산을 수행하는 기호
- 피연산자(operand): 연산의 작업 대상

#### ◆ 종류

- ❖ 산술 연산자(+,-,\* , /, %, //,\*\*), 할당 연산자(=, +=, -=, \*=, /=)
- ❖ 비교 연산자(>,<,==, !=,>=,<=) 논리 연산자(and, or, not)
- ❖ 비트 연산자(&, |, ^, ~,<<,>>), 멤버 연산자(in , not in)
- ❖ 식별 연산자(is, is not)

### 3.3.1 기본 연산자 및 우선순위

〈표 3.2〉 파이썬 연산자의 종류 및 우선순위

연산자	명칭	설명	예시 → 결과	우선 순위
[], {}, ()	괄호	리스트, 사전, 튜플 생성	a = [1, 2, 3]	1
[], ()	첨자	리스트, 튜플의 원소 참조	a[1]	2
**	거듭제곱	앞 피연산자를 뒤 피연산자만큼 곱함	$3^{**}3 \rightarrow 27$	3
~	단항	보수를 취함		4
-	マイ너스	부호 음수로 바꿈		
*	곱셈	두 연산자를 곱함	$5^*3 \rightarrow 15$	5
/	나눗셈	앞 피연산자를 뒤 피연산자로 나눔	$5/2 \rightarrow 2.5$	
//	나눗셈몫	나누어서 몫만 가져옴, 결과 정수형	$5//2 \rightarrow 2$	
%	나머지	나누어서 나머지만 가져옴, 결과 정수형	$5\%2 \rightarrow 1$	
+	덧셈	피연산자 숫자일 경우: 두 수를 더함 피연산자 문자일 경우: 두 문자를 붙임	$5+6 \rightarrow 11$ $'a' + 'b' \rightarrow 'ab'$	6
-	뺄셈	앞 연산자에서 뒤 연산자 뺄	$5 - 3 \rightarrow 2$	

### 3.3.1 기본 연산자 및 우선순위

$>$ , $>=$	초과, 이상	앞 연산자가 뒤 연산자 초과(이상)인지 검사	$5 > 3 \rightarrow \text{True}$	7
$<$ , $<=$	미만, 이하	앞 연산자가 뒤 연산자 미만(이하)인지 검사	$5 < 3 \rightarrow \text{False}$	
$=$	같음	앞과 뒤 연산자의 값이 같은지 검사	$5 == 3 \rightarrow \text{False}$	8
$\neq$ , $\neq$	같지 않음	앞과 뒤 연산자의 값이 다르면 참으로 평가	$5 \neq 3 \rightarrow \text{True}$	
in, not in	멤버	객체 내에 원소의 존재 여부 검사	$[1,4,5] \text{ in } 4 \rightarrow \text{True}$	9
is, is not	식별	같은 객체 인지 검사	$5 \text{ is } 4 \rightarrow \text{False}$	
and	논리곱	두 피연산자 결과가 모두 참인지 검사	$(5 > 3) \text{ and } (3 < 2) \rightarrow \text{False}$	10
or	논리합	두 피연산자 결과가 하나라도 참인지 검사	$(5 > 3) \text{ or } (3 < 2) \rightarrow \text{True}$	
not	부정	뒤 피연산자 결과에 반대	$\text{not}(5 > 3) \rightarrow \text{False}$	11
=	할당	앞 피연산자에 뒤 피연산자 결과 저장	$a = 5$	12

### 3.3.2 슬라이스(:) 연산자

- ❖ 열거형 객체의 일부를 잘라서 가져오는 새 객체 생성

**열거형 객체[시작 인덱스:종료 인덱스:인덱스 증가폭]**

- ❖ 종료 인덱스는 범위에 포함되지 않음
- ❖ 인덱스 증가폭
  - 범위 내에서 잘라서 가져올 때 인덱스를 건너뛰는 폭
  - 기본값은 1씩 증가하며, 음수는 감소를 의미

### 3.3 연산자

#### ◆ 예제

##### 예제 3.3.1 슬라이스 연산자 - 04.slice

```
01 a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
02 print('a = ', a)
03 print('a[:2] ->', a[:2])                                # 0~3 범위
04 print('a[4:-1] ->', a[4:-1])                            # 4~8 범위 (-1은 마지막 인덱스)
05 print('a[2::2] ->', a[2::2])                            # 2~9 범위, 2씩 증가
06 print('a[::-1] ->', a[::-1])                            # 전체범위 역순 - 자주 사용됨
07 print('a[1::-1] ->', a[1::-1])                            # 첫 2개 원소 역순 - 자주 사용됨
08 print('a[7:1:-2] ->', a[7:1:-2])                      # 7~2 범위 2씩 감소 - 역순
09 print('a[:-4:-1] ->', a[:-4:-1])                      # 9~7 범위 1씩 감소 - 역순
```

Run: 04.slice\_operate

```
C:\Python\python.exe D:/source/chap03/04.slice_operate.py
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a[:2] -> [0, 1]
a[4:-1] -> [4, 5, 6, 7, 8]
a[2::2] -> [2, 4, 6, 8]
a[::-1] -> [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
a[1::-1] -> [1, 0]
a[7:1:-2] -> [7, 5, 3]
a[:-4:-1] -> [9, 8, 7]
```

## 3.4 기본 명령문

- ◆ 3.4.1 조건문
- ◆ 3.4.2 반복하기
- ◆ 3.4.3 순회하기

## 3.4.1 조건문

### ◆ 조건문

- ❖ IF 다음에 조건 지정, 조건 뒤에 콜론(:) 추가
- ❖ 조건은 비교 연산자(>,<, ==, >=, <=) 사용
- ❖ 논리 연산자(AND, OR)로 비교 연산자 여러 번 사용 가능
- ❖ ELIF 문은 여러 번 반복하여 추가 가능
- ❖ ELIF 문과 ELSE 문은 필요 없는 경우 생략 가능
- ❖ 형식에 맞게 들여쓰기(INDENTATION) 필수

`if` 조건1:

명령문들

`elif` 조건2:

명령문들

`else:`

명령문들

## 3.4.1 조건문

### ◆ 조건문

#### ❖ 예제

예제 3.4.1

### 윤년과 평년 검사 - 05.if.py

```
01 year = 2020
02
03 if (year % 4==0) and (year % 100 != 0):      # 4년마다 윤년, 100년마다 윤년 아님
04     print(year, "는 윤년입니다.")
05 elif year % 400==0:                            # 400년마다 윤년
06     print(year, "는 윤년입니다.")
07 else:
08     print(year, "는 윤년이 아닙니다.")
```

Run: 05.if  
C:\Python\python.exe D:/source/chap03/05. if. py  
2020 는 윤년입니다.

## 3.4.2 반복하기

### ◆ 컴퓨터가 잘하는 것 세 개

- ❖ 기억하기 - 변수가 담당
- ❖ 계산하기 - 다양한 연산자와 함수
- ❖ 반복하기 – while 문 , for 문

### ◆ while 문

- ❖ 특정 조건이 참일 경우, 계속해서 블록(들여쓰기 되어있는 부분)의 명령문들을 반복 실행
- ❖ 조건 뒤에 콜론(:) 추가

`while` 조건:

명령문들

`else:`

명령문들

## 3.4.2 반복하기

### ◆ 예제

예제 3.4.2

4개 정수 입력받기 – 06.while.p

```
01 n=3
02 while n>=0:
03     m = input("Enter a integer: ")
04     if int(m)==0: break
05     n = n - 1
06 else:
07     print('4 inputs.')
```

Run: 06.while ▾

C:\Python\python.exe D:\source\chap03\06.while

Enter a interger: 1  
Enter a interger: 2  
Enter a interger: 3  
Enter a interger: 4  
4 inputs.

### 3.4.3 순회하기

#### ◆ for ~ in 문

- ❖ 열거형(sequence) 객체의 원소 순회
- ❖ 블록 내의 명령들을 반복하여 실행할 때 사용되는 명령문

#### ◆ 형식

```
for 변수 in 리스트:
```

```
    명령문 1
```

```
    명령문 2
```

```
....
```

- ❖ 리스트 대신 튜플, 사전, 문자열 가능

- ❖ range() 함수, enumerate() 함수, zip() 함수로 객체 생성 후 순회 가능

## 3.4.3 순회하기

### ◆ 예제

#### 예제 3.4.3 리스트 원소합 구하기 – 07.for.py

```
01 kor = [70, 80 ,90, 40, 50]          # 리스트 선언
02 eng = [90, 80 ,70, 70, 60]
03 sum1, sum2, sum3, sum4 = 0, 0, 0, 0   # 누적 변수 초기화
04
05 for i in range(0, 5):                # range() 함수로 범위지정
06     sum1 = sum1 + kor[i] + eng[i]
07
08 for k in kor:                        # 리스트 원소 순회
09     sum2 = sum2 + k
10 for e in eng:
11     sum2 = sum2 + e
12
13 for i, k in enumerate(kor):          # 리스트의 인덱스와 원소로 순회
14     sum3 = sum3 + k + eng[i]
15
16 for k, e in zip(kor, eng):           # 여러 객체 동시 순회
17     sum4 = sum4 + k + e
18
19 print('sum1=', sum1), print('sum2=', sum2)
20 print('sum3=', sum3), print('sum4=', sum4)
```

Run: 07.for ▾  
C:\Python\python.exe D:/source/chap03/07.for.py  
sum1= 700  
sum2= 700  
sum3= 700  
sum4= 700

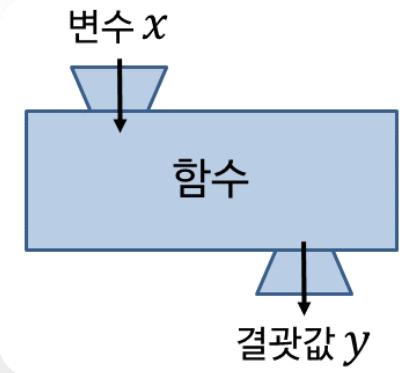
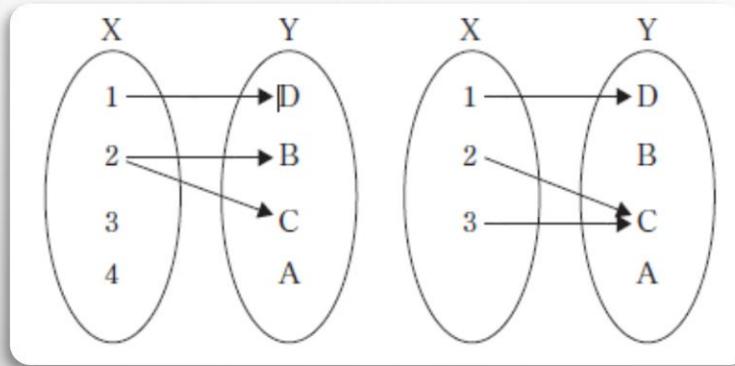
## 3.5 함수와 라이브러리

- ◆ 3.5.1 함수
- ◆ 3.5.2 모듈(Module), 패키지
- ◆ 3.5.3 파이썬 내장함수

## 3.5.1 함수

### ◆ 함수(function)란

- ❖ 첫 번째 집합(정의역)의 임의의 한 원소를 두 번째 집합(공역)의 오직 한 원소에 대응시키는 이항 관계



### ◆ 컴퓨터 언어에서 함수란

- ❖ 일정한 작업을 수행하는 코드 블록
- ❖ 보통 반복적으로 계속 사용되는 코드들을 함수로 정의하여 사용
- ❖ 서브루틴(subroutine), 루틴(routine), 메서드(method), 프로시저(procedure)

## 3.5.1 함수

### ◆ 형식

- ❖ def 키워드 사용
- ❖ 함수명과 인수로 구성
- ❖ Return 키워드로 결과 반환

```
def 함수명(인수명1, 인수명2, ... ) :  
    수행할 내용1  
    수행할 내용2  
    수행할 내용3  
    return 반환값1, 반환값2, ..
```

## 3.5.1 함수

### 예제 3.5.1 도형 넓이 구하기 - 08.def.py

```
01 def calc_area(type, a, b, c=None):
02     if type == 1 :                                # 사각형
03         result = a * b
04         msg = '사각형'
05     elif type == 2:                               # 삼각형
06         result = a * b / 2
07         msg = '삼각형'
08     elif type == 3:                               # 평행사변형
09         result = (a + b) * c / 2
10         msg = '평행사변형'
11     return result, msg
12
13 def say():                                     # 인수, 반환값
14     print('넓이를 구해요')
15
16 def write(result, msg):                         # 반환값
17     print( msg, '넓이는', result , 'm2 입니다.' )
```

Run: 08.def

```
C:\Python\python.exe D:/source/chap03/08.def.py
넓이를 구해요
<class 'tuple'>
<class 'float'> <class 'str'>
사각형 넓이는 25 m2 입니다.
삼각형 넓이는 12.5 m2 입니다.
평행사변형 넓이는 42.5 m2 입니다.
```

```
19 say()                                         # 함수 호출 - 인수
20 ret = calc_area(type=1, a=5, b=5)           # 함수 호출 - 튜플
21 area, msg = calc_area(2, 5, 5)             # 함수 호출 - 튜플
22 area2, _ = calc_area(3, 10, 7, 5)          # 함수 호출 - 반환값
23
24 print(type(ret))                           # 인수, 반환값
25 print(type(area), type(msg) )              # 인수, 반환값
26 write(ret[0], ret[1])                      # 반환값
27 write(area, msg)
28 write(area2, '평행사변형' )
```

## 3.5.2 모듈(Module), 패키지

### ◆ 컴퓨터 언어에서 모듈의 의미

- ❖ 함수들의 모음, 클래스들의 모음, 라이브러리, 프로그램들 등등 각기 정의에 따라 조금씩 다름

### ◆ 파이썬에서 모듈

- ❖ 함수나 변수 또는 클래스를 모아 놓은 파일
- ❖ 다른 파이썬 프로그램에서 불러와 사용할 수 있게 만든 파일
- ❖ 장점
  - 매번 함수를 구현하지 않고 불러와 사용
  - 소스 코드도 간결, 이해 쉬움, 매우 편리

## 3.5.2 모듈(Module), 패키지

### ◆ 3.5.1 예제의 함수들로 파일 생성

#### 예제 3.5.2 헤더 파일 header\_area.py

```
01 def calc_area(type, a, b, c=None): ...
12
13 def say(): ...                                # 인수, 반환값 없는 함수 구현
15
16 def write(result, msg): ...                    # 반환값 있는 함수 구현
```

Run: 09.module

```
C:\Python\python.exe D:/source/chap03/09.module
넓이를 구해요
사각형 넓이는 25 m2 입니다.
```

### ◆ 위 함수 사용하기

#### 예제 3.5.2 모듈 임포트하기 09.module.py

```
01 import chap03.header_area as mod           # header_area.py 파일을 모듈로 임포트
02 from chap03.header_area import write        # header_area.py 파일내 write 함수만 임포트
03
04 mod.say()                                    # area 모듈내 함수 호출
05 area, msg= mod.calc_area(type=1, a=5, b=5)  # area 모듈내 함수 호출
06 write(area, msg)
```

### 3.5.3 파이썬 내장함수

#### ◆파이썬 내장함수

##### ❖ 파이썬에서 미리 만들어 놓은 기본적인 함수들

함수	설명	함수	설명
abs(a)	a의 절댓값 반환	max(a)	객체 a에서 최대 원소값 반환
chr(a)	a 값을 문자 반환	min(a)	객체 a에서 최소 원소값 반환
divmod(a,b)	나눈 몫과 나머지 반환	open()	파일 읽기위한 파일 객체 만들기
enumerate(a)	객체 a의 인덱스와 원소 반환	ord(a)	문자 a의 아스키코드 값 반환
eval(a)	문자열 a를 실행	pow(a, b)	a를 b 제곱한 결과 반환
input()	키보드로 값을 입력받는 함수	range(a,b,c)	c 간격 갖는 범위(a~b)의 객체 반환
int(a)	a를 정수형으로 변환	round(a)	a를 반올림하여 반환
isinstance(a)	객체 a가 클래스의 인스턴스 인지 검사	str(a)	a를 문자열로 반환
len(a)	객체 a의 원소 개수 반환	sum(a)	객체 a 원소값들의 합 반환
list(a)	객체 a를 리스트로 변환	tuple(a)	객체 a를 튜플로 변환
map(int, a)	객체 a의 원소들을 함수 int로 수행한 결과 반환	type(a)	a의 자료형 반환
print()	콘솔창에 결과 출력	zip(a,b ,,,)	여러 객체를 묶어 주는 함수

### 3.5.3 파이썬 내장함수

#### ◆ 내장 함수 예제

예제 3.5.3

내장함수 예제 10.inner\_fuction.py

```
01 a = [1.5, 2, 3, 4, 5]           # 리스트 생성
02 b = map(float, a)                # 실수 변환
03 c = divmod(5, 3)                # 몫&나머지
04
05 print('최댓값: ', max(a), ' 최솟값: ', min(a) )
06 print('몫과 나머지: ', c )
07 print('c의 자료형: ', type(c), type(c[0]), type(c[1]) )
08
09 print('2의 4제곱: ', pow(2, 4) )
10 print('절댓값: ', abs(-4) )
```

Run: 10.inner\_fuction

```
C:\Python\python.exe D:/source/chap03/10.inner_fuction.py
최댓값: 5 최솟값: 1.5
몫과 나머지: (1, 2)
c의 자료형: <class 'tuple'> <class 'int'> <class 'int'>
2의 4제곱: 16
절댓값: 4
```

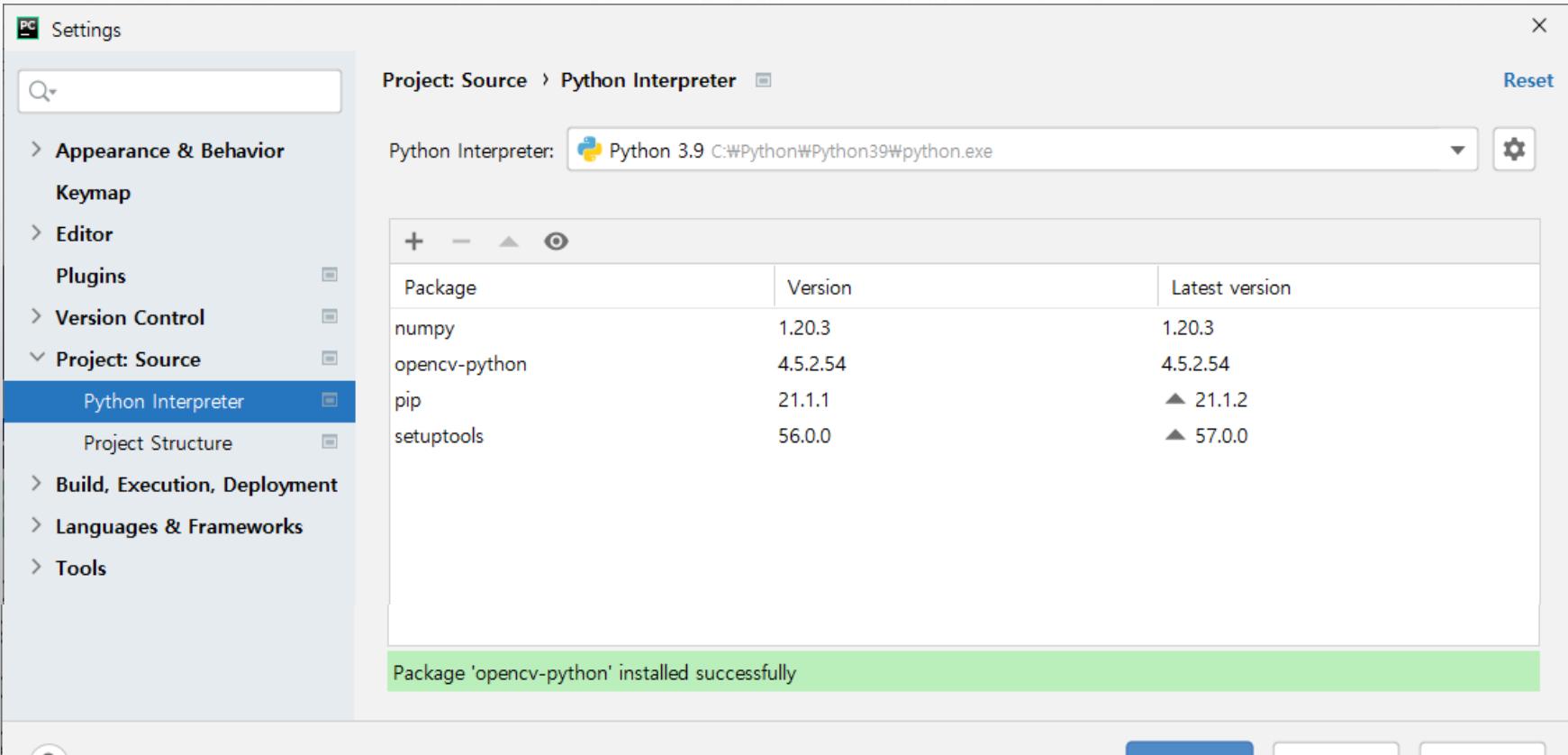
## 3.6 넘파이(numpy) 패키지

- ◆ OpenCV 라이브러리를 사용한 영상처리
  - ❖ 영상처리는 2차원 데이터의 처리가 기본
  - ❖ 파이썬에는 넘파이(numpy)라는 다차원 데이터를 처리하는 라이브러리 지원
  - ❖ OpenCV 함수들도 넘파이 객체를 기본 데이터 구조(입력 인수, 반환 객체)로 사용
  
- ◆ 파이참에 넘파이 라이브러리 추가 방법
  - ❖ [Settings] 윈도우 → [Project interpreter]에서 라이브러리 추가

# 3.6 넘파이(numpy) 패키지

## ◆ 설정 대화상자 열기

- ❖ 프로젝트 소스 → 파이썬 인터프리터 →



# 3.6 넘파이(numpy) 패키지

## 예제 3.6.1 넘파이 자료형 – 11.numpy.py

```
01 import numpy as np
02
03 list1, list2 = [1, 2, 3] , [4, 5.0, 6]
04 a, b = np.array(list1), np.array(list2)
05
06 c = a + b
07 d = a - b
08 e = a * b
09 f = a / b
10 g = a * 2
11 h = b + 2
12
13 print('a 자료형:', type(a), type(a[0]))
14 print('b 자료형:', type(b), type(b[0]))
15 print('c 자료형:', type(c), type(c[0]))
16 print('g 자료형:', type(g), type(g[0]))
17 print(c, d, e)
18 print(f, g, h)
```

Run: 11.numpy ▾

```
C:\Python\python.exe D:/source/chap03/11.numpy.py
a 자료형: <class 'numpy.ndarray'> <class 'numpy.int32'>
b 자료형: <class 'numpy.ndarray'> <class 'numpy.float64'>
c 자료형: <class 'numpy.ndarray'> <class 'numpy.float64'>
g 자료형: <class 'numpy.ndarray'> <class 'numpy.int32'>
[5. 7. 9.] [-3. -3. -3.] [ 4. 10. 18.]
[0.25 0.4 0.5] [2 4 6] [6. 7. 8.]
```

# 리스트로 ndarray 객체 생성

# ndarray 객체 연산 – np.add(list1, list2)

# np.subtract(list1, list2)

# np.multiply(list1, list2)

# np.divide(list1, list2)

# 스칼라 곱

# 스칼라 합

# 결과 출력

## 3.6 넘파이(numpy) 패키지

## ◆ 예제

### 예제 3.6.2 행렬 원소로 지정값 생성하기 – 12.numpy2.py

```
01 import numpy as np          # 넘파이 모듈 임포트
02
03 a = np.zeros((2, 5), np.int)  # 원소값 0 행렬 - 2행, 5열, 32비트 정수형
04 b = np.ones((3, 1), np.uint8) # 원소값 1 행렬 - 3행, 1열, 부호없는 8비트 정수형
05 c = np.empty((1, 5), np.float) # 값없음 행렬 1행, 5열,
06 d = np.full(5, 15, np.float32) # 원소값 15, 1차원 행렬, 32비트 실수형
07
08 print(type(a), type(a[0]), type(a[0][0])) # 객체 자료형, 객체 원소의 자료형 출력
09 print(type(b), type(b[0]), type(b[0][0]))
10 print(type(c), type(c[0]), type(c[0][0]))
11 print(type(d), type(d[0]))
12 print('c 형태:', c.shape, ' d 형태:', d.shape) # 객체 형태 출력
13 print(a), print(b) # 객체 원소 출력
14 print(c), print(d)
```

# 3.6 넘파이(numpy) 패키지

## ◆ 실행결과

```
Run: 12.numpy2 ▾
```

```
C:\Python\python.exe D:/source/chap03/12.numpy2.py
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.int32'>
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.uint8'>
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.float64'>
<class 'numpy.ndarray'> <class 'numpy.float32'>
c 형태: (1, 5)      d 형태: (5,) → 1차원 배열 형태
[[0 0 0 0 0]]        → 2차원 배열 형태
[[1]
 [1]
 [1]]                → 2차원 배열(3행, 1열)
[[0.000e+000 0.000e+000 0.000e+000 3.063e-321 0.000e+000]] → 2차원 배열(1행, 5열)
[15. 15. 15. 15. 15.] → 1차원 배열
```

# 3.6 넘파이(numpy) 패키지

예제 3.6.3

행렬 원소로 임의값 생성하기 – 13.nun

```
01 import numpy as np
02
03 np.random.seed(10)
04 a = np.random.rand(2, 3)
05 b = np.random.randn(3, 2)
06 c = np.random.rand(6)
07 d = np.random.randint(1, 100, 6)
08 c = np.reshape(c, (2, 3))
09 d = d.reshape(2, -1)
10
11 print('a 형태:', a.shape, '\n', a)
12 print('b 형태:', b.shape, '\n', b)
13 print('c 형태:', c.shape, '\n', c)
14 print('d 형태:', d.shape, '\n', d)
15
16 print('다차원 객체 1차원 변환 방법')
17 print('a =', a.flatten())
18 print('b =', np.ravel(b))
19 print('c =', np.reshape(c, (-1, )))
20 print('d =', d.reshape(-1, ))
```

Run: 12.numpy3

```
C:\Python\python.exe D:/source/chap03/12.numpy3.py
a 형태: (2, 3)
[[0.77132064 0.02075195 0.63364823]
 [0.74880388 0.49850701 0.22479665]]
b 형태: (3, 2)
[[0.62133597 -0.72008556]
 [0.26551159 0.10854853]
 [0.00429143 -0.17460021]]
c 형태: (2, 3)
[[0.81262096 0.61252607 0.72175532]
 [0.29187607 0.91777412 0.71457578]]
d 형태: (2, 3)
[[14 26 14]
 [93 87 31]]
```

다차원 객체 1차원 변환 방법

```
a = [0.77132064 0.02075195 0.63364823 0.74880388 0.49850701 0.
b = [0.62133597 -0.72008556 0.26551159 0.10854853 0.00429143 -0.17460021]
c = [0.81262096 0.61252607 0.72175532 0.29187607 0.91777412 0.71457578]
d = [14 26 14 93 87 31]
```

# 다차원 ndarray 객체를 1차원 벡터로 변환

# 다차원 모든 객체를 1차원 벡터로 변환

# 넘파이의 reshape() 함수 사용

# ndarray 객체 내장 reshape() 함수 사용

# 단원 요약

- ◆ 파이썬에서 리터럴은 숫자나 문자열과 같은 값 그 자체를 의미한다.
  - ❖ 상수는 문자와 숫자가 저장된 변수를 변경하지 못하도록 지정하는 것(대문자를 변수로 지정해 사용)
- ◆ 변수는 리터럴이 저장된 기억 장치의 특정 주소를 의미하며, 이 주소에 저장된 리터럴은 다양한 방법으로 변경할 수 있기 때문에 변수라 명명한다.
- ◆ 파이썬에서 수식은 연산자와 피연산자로 구성되며, 피연산자에는 리터럴, 변수, 함수 등이 지정되며, 연산자에는 사칙연산자, 논리 연산자, 비교 연산자 등 다양하게 있다.
- ◆ 슬라이스 연산자는 열거형(sequence) 객체의 특정 원소들을 잘라서 가져올 때 사용된다. 시작 인덱스, 종료 인덱스, 간격이 대괄호로 묶어서 구성된다.
- ◆ if 문은 조건을 주어 참이면 조건 이후의 명령들을 수행하며, 그렇지 않으면 else 이후의 명령 문들을 수행하여 조건에 따라서 다른 명령을 적용하고자 할 때 사용한다.

# 단원 요약

- ◆ while 문과 for 문은 블록내의 명령들의 수행을 반복할 때 사용하는 명령문이다.
  - ❖ while문은 조건에 맞을 경우에 반복 수행
  - ❖ for 문은 열거형 객체의 원소들을 순회하는 동안 반복 수행
- ◆ 함수(function)는 일정한 작업을 수행하는 코드 블럭으로 반복적으로 계속 사용되는 코드들을 함수로 정의하여 사용한다.
  - ❖ 파이썬에서 함수는 def 키워드를 사용하여 정의
- ◆ 모듈은 자주 사용하는 함수들을 모아서 하나의 파일로 구성하여 다른 파일에서 참조하여 사용하는 것을 말한다.
  - ❖ import 문을 이용해서 필요한 파일명과 함수명으로 함수를 참조할 수 있다.
- ◆ 넘파이 패키지는 다차원 배열을 처리하는데 유용한 함수 제공한다.
  - ❖ OpenCV의 많은 함수들이 넘파이(numpy)의 ndarray객체를 기본 데이터 구조(입력 인수, 반환 객체)로 사용한다.