# 02. 파이썬 기초 익히기

#### 학습목표

데이터 분석을 하려면 알아야 하는 파이썬 기본 문법과 함수를 최소한으로 소개합니다. 프로그래밍 언어를 눈으로만 익히면 실제로 사용할 수 없습니다. 가능하면 코드를 타이핑하면서 공부하세요. 그러면 파이썬과 조금 더 친숙해 질 겁니다. 파이썬에 익숙한 분은 이 장을 건너뛰어 3장으로 이동하세요.

#### 파이썬

1991년 귀도 반 로섬이 발표한 고급 프로그래밍 언어입니다. 문법이 간단하고 라이브러리가 풍부합니다. 인터프리터식, 객체지향적, 동적 타이핑dynamically typed 대화형 언어로서 백엔드와 머신러닝 분야를 비롯한 다양한 영역에서 사용됩니다. 프로그래밍 언어 사용 랭킹을 제공하는 티오베 발표에 따르면 2022년 1월 파이썬은 1위를 차지했습니다.[[1]](#footnote-0) 백엔드와 머신러닝 분야를 비롯한 다양한 영역에서 활발히 사용됩니다.

#### 학습 순서



## 2.1 프로그래밍 기본 : 산술 연산, 변수, 출력

파이썬 프로그램을 작성하려면 기본 지식이 있어야 합니다. 연산, 변수, 출력 개념을 알아야 합니다. 간단히 알아봅시다.

#### 산술 연산

파이썬이 지원하는 산술 연산자는 다음과 같이 7가지입니다. 더하기/빼기/곱하기/나누기는 수학과 같습니다. 거듭제곱은 \*\*, 나누기 연산 후 소수점 이하 수를 버린 정수는 //, 나누기 연산 후 나머지만 구하는 %만 알아두면 파이썬으로 수를 계산하는 데 문제가 없을 겁니다.

▽ 산술 연산자

| **연산자** | **설명** | **예문** |
| --- | --- | --- |
| + | 더하기 | 수식 : 3+2  결과 : 5 |
| **-** | 빼기 | 수식 : 3-2  결과 : 1 |
| \* | 곱하기 | 수식 : 3\*2  결과 : 6 |
| \*\* | 거듭제곱 | 수식 : 3\*\*2  결과 : 9 |
| / | 나누기 | 수식 : 3/2  결과 : 1.5 |
| // | 나누기 연산 후 소수점 이하 수를 버린 정수 | 수식 : 3//2  결과 : 1 |
| % | 나누기 연산 후 나머지 | 수식 : 3%2  결과 : 1 |

#### 변수

변수는 값을 담는 공간입니다. 산술 연산에 사용할 a와 b에 각각 3과 2를 넣어 산술 연산을 해보겠습니다.

<코드/>

a = 3

b = 2

a+b

</>

<출력 결과/>

5

</>

a와 b를 더한 결과인 5가 출력되었습니다. 여기서는 a+b만 사용해도 원하는 결과를 볼 수 있습니다만, 경우에 따라 별도의 출력 함수를 사용해야 보기 좋은 출력 결과를 얻을 수 있습니다.

#### 출력 함수

파이썬에서는 print() 함수를 사용해 출력을 합니다. 앞의 코드에 print() 함수를 적용해보겠습니다.

<코드/>

a = 3

b = 2

print(a+b)

</>

<출력 결과/>

5

</>

print()를 사용하지 않았을 때와 같은 결과가 나왔습니다. 대부분의 경우 print()를 사용하지 않아도 동일한 결과를 볼 수 있지만, 종종 print()를 써야만 가독성 있는 결과물이 나오므로 숙지해두는 것이 좋습니다.

## 2.2 자료형과 자료구조

파이썬은 크게 숫자형, 논리형, 군집형을 제공합니다. 사칙연산과 같은 계산을 할 수 있는 숫자형으로는 정수형, 실수형, 복소수형이 있습니다. 하나 이상의 값을 가질 수 있는 군집형으로는 문자열, 딕셔너리, 리스트, 세트, 튜플이 있습니다.

▽ 파이썬 자료형



### 2.2.1 숫자형

숫자와 관련된 자료형으로, 기본적인 연산이 가능한 데이터입니다.

▽ 숫자형

| 유형 | 설명 | 예 |
| --- | --- | --- |
| 정수형  (Integer) | 소수점이 없는 숫자 | 1, 100 |
| 실수형  (Float) | 소수점이 있는 숫자 | 1.1, 3.14 |
| 복소수형 (Complex) | 실수와 허수로 이루어진 숫자 | 2 + 5j |

### 2.2.2 논리형

논리형 자료형은 불리언/불린Boolean 혹은 부울Bool이라고 불립니다. True는 참, False는 거짓을 나타냅니다. 따옴표 없이 True와 False를 입력해 정의할 수 있고, 아래와 같이 비교연산을 하면 불리언값이 결과로 나타납니다.

| 3 > 1 |
| --- |

True

| 3 > 7 |
| --- |

False

### 2.2.3 군집형

하나 이상의 값을 가질 수 있는 자료형으로 리스트, 튜플, 딕셔너리, 세트, 문자열이 있습니다.

▽ 군집형

| 유형 | 설명 | 순서 | 수정 | 생성자 | 예문 |
| --- | --- | --- | --- | --- | --- |
| 리스트 | 순서가 있는 데이터의 모음으로, 정의된 리스트라도 후에 삭제, 추가 등 변경할 수 있습니다. | O | O | [ ]  or  list() | [1, 2, 3] |
| 튜플 | 리스트와 마찬가지로 순서가 있는 데이터이나, 한 번 정의되면 수정이 불가능합니다. | O | X | ( )  or  tuple() | (4, 5, 6) |
| 딕셔너리 | 키와 값으로 구성된 자료구조로, 하나의 키에 여러 가지 값들을 매핑할 수도 있습니다. 또한 딕셔너리에서는 키를 사용해 그에 해당하는 값들을 쉽게 불러올 수 있습니다. | X | O | { }  or  dict() | {'name': 'Sam', 'age': 20} |
| 세트 | 중복된 값을 보유하지 않는 자료구조로, 리스트나 튜플과는 달리 순서가 없는 형태입니다. | X | O | { }  or  set() | {1,3,5} |
| 문자열 | 따옴표 안에 들어 있는 자료 형태를 말하며, 문자형이기 때문에 숫자형같은 사칙연산은 불가능합니다. 숫자 데이터라도 ‘10’과 같이 따옴표 안에 들어 있으면 숫자형이 아닌 문자형으로 인식됩니다. | O | X | ' '  or  str() | 'string' |

#### 리스트와 튜플

리스트는 데이터들의 일련의 배열입니다. [ ]를 사용해 정의하며, 1~10까지 숫자를 담은 리스트를 다음과 같이 만들 수 있습니다.

| [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |
| --- |

위의 리스트를 sample\_list라는 변수에 저장하고 sample\_list를 실행하면 다음과 같이 저장된 리스트를 확인할 수 있습니다.

| sample\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] sample\_list |
| --- |

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

튜플은 리스트와 매우 흡사하며 ( )으로 정의합니다. 1~10까지 숫자를 담은 튜플을 만들어서 sample\_tuple에 저장해봅시다.

| sample\_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) sample\_tuple |
| --- |

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

리스트와 튜플에서는 특정 자리의 값들을 가져오는 인덱싱이 가능합니다. sample\_list에서 첫 번째 자리(인덱스 0)와, sample\_tuple에서는 네 번째 자리(인덱스 3)를 인덱싱해보겠습니다.

<warning/>

파이썬에서는 첫 번째 인덱싱 위치를 1이 아닌 0으로 표기하므로, 혼동하지 않도록 주의합시다.

</>

| sample\_list[0] |
| --- |

1

| sample\_tuple[3] |
| --- |

4

파이썬 자료형에서 ‘순서가 있다'는 말은 각 값마다 특정 위치가 정해져 있다는 의미입니다. 그래서 위칫값(인덱스)을 지정해 해당 위치의 값을 불러올 수 있습니다. 순서가 없는 데이터의 예는 잠시 후 딕셔너리와 세트에서 살펴보겠습니다.  
  
리스트는 저장된 값들을 수정/삭제할 수 있는 반면, 튜플은 정의된 이후에 변경이 불가능하다는 결정적인 차이가 있습니다. 리스트에서 특정값을 제거하는 데 remove() 함수를 사용합니다. sample\_list에서 10을 제거해봅시다.

| sample\_list.remove(10) sample\_list |
| --- |

[1, 2, 3, 4, 5, 6, 7, 8, 9]

이번에는 sample\_tuple에서 10을 제거하겠습니다.

| sample\_tuple.remove(10) |
| --- |

**---------------------------------------------------------------------------**

**AttributeError** Traceback (most recent call last)

**<ipython-input-6-5e9ff8210e66>** in <module>

**----> 1** sample\_tuple**.**remove**(10)**

**AttributeError**: 'tuple' object has no attribute 'remove'

튜플에서는 에러가 발생합니다. 이처럼 한 번 정의된 튜플은 변경이 불가합니다.

리스트에서 유용하게 쓰일 함수 중 하나로 append() 함수를 소개하겠습니다. append()는 리스트에 새로운 값을 추가할 때 사용합니다. sample\_list에 100을 추가하겠습니다.

| sample\_list.append(100) sample\_list |
| --- |

[1, 2, 3, 4, 5, 6, 7, 8, 9, 100]

#### 딕셔너리

딕셔너리는 키key와 값value의 조합으로 이루어졌습니다. { }로 정의하며 키와 값은 :로 구분해줍니다. 여러 쌍의 키/값을 넣을 수 있으며 ,(쉼표)로 구분해주면 됩니다. 아래의 예시 데이터를 딕셔너리 형태로 만들어봅시다.

| Name | Age |
| --- | --- |
| John | 23 |
| Ann | 22 |
| Kevin | 21 |

| sample\_dic = {'name': ['John','Ann','Kevin'], 'age': [23,22,21]} sample\_dic |
| --- |

{'name': ['John', 'Ann', 'Kevin'], 'age': [23, 22, 21]}

하나의 값value에 여러 데이터를 넣어줄 때는 위의 예시처럼 리스트(혹은 튜플)를 이용해주면 됩니다.   
  
그리고 딕셔너리는 정렬된 형태의 데이터가 아니기 때문에, 특정 위치의 숫자를 지정하여 데이터를 불러올 수 없습니다. 하지만 딕셔너리는 키를 가지고 있기 때문에, 특정 키를 [ ] 안에 넣어주면 해당 키에 해당하는 값들만 불러올 수 있습니다. sample\_dic에서 name 키에 해당하는 값들만 불러봅시다.

| sample\_dic['name'] |
| --- |

['John', 'Ann', 'Kevin']

#### 세트

세트는 수학에서의 ‘집합’을 의미합니다. 세트는 유용한 자료형이지만, 이 책에서는 사용할 일이 없기 때문에 여기서는 가볍게 소개만 하고 넘어가겠습니다. 세트는 중복되는 값이 없도록 표현되는 특징이 있습니다. 실습을 위해, 아래와 같이 3이 여러 번 포함된 리스트를 만들어보겠습니다.

| set\_list = [1,2,3,3,4,5,3,3] |
| --- |

이 리스트를 세트 형식으로 변경해보겠습니다.

| set(set\_list) |
| --- |

{1, 2, 3, 4, 5}

보시는 바처럼 4개였던 3이 하나로 줄었습니다.

세트 역시는 순서가 있는 데이터가 아니기 때문에 인덱싱이 불가능합니다. 실습을 위해, 위에서 정의한 세트를 sample\_set이라는 이름으로 저장해봅시다.

| sample\_set = set(set\_list) |
| --- |

리스트에서처럼 [0]을 사용하여 첫 번째자리를 인덱싱해보겠습니다.

| sample\_set[0] |
| --- |

**---------------------------------------------------------------------------**

**TypeError** Traceback (most recent call last)

**<ipython-input-34-aac88c932db3>** in <module>

**----> 1** sample\_set**[0]**

**TypeError**: 'set' object is not subscriptable

에러가 발생합니다. 또한 세트는 딕셔너리와 달리 키도 없기 때문에 [ ]를 사용한 인덱싱도 불가능합니다.

마지막으로 세트는 수정이 가능합니다. add() 함수를 사용해 추가할 수 있습니다(리스트에서는 append() 함수를 사용했습니다). sample\_set에 6을 추가하고 결과를 확인해보겠습니다.

| sample\_set.add(6) sample\_set |
| --- |

{1, 2, 3, 4, 5, 6}

## 2.3 반복문 : for문, while문

for문과 while문은 코드를 반복시킵니다. for문은 대상 리스트의 모든 원소가 반환될 때까지, while문은 조건식이 참인 동안 반복합니다.

#### for문

for문 원형은 다음과 같습니다.

| for i in {대상 리스트}:  {원하는 연산} |
| --- |

위와 같이 작성하면 **대상 리스트**에 있는 값들이 하나하나 i에 대입됩니다.

예를 들어 1~3까지의 숫자들에 각각 5를 더하는 코드를 반복문 없이 다음과 같이 만들 수 있습니다.

<코드/>

1+5

2+5

3+5

</>

for문으로 구현하면 이렇습니다.

| for i in [1, 2, 3]:  print(i+5) |
| --- |

6

7

8

대상 리스트인 [1, 2, 3]의 숫자들이 순서대로 하나씩 i에 대입되어 print(i+5) 연산이 수행됩니다. 대상 리스트의 길이가 3이기 때문에 for 구문이 총 3번 반복됩니다. 3번 반복이라 코드가 더 간단하다는 생각이 안 드는군요. 하지만 100번 반복일 때를 생각해보세요. 훨씬 간단할 겁니다. 게다가 for문을 사용하면 유지보수도 더 쉬워지죠.

#### while문

while문은 for문과 비슷합니다. 차이가 있다면 특정 조건이 되면 반복을 멈춘다는 겁니다. while문 원형은 다음과 같습니다.

| while {조건}:  {원하는 연산} |
| --- |

여기서 조건을 만족시키면 계속 while문이 돌아가고, 조건을 만족하지 않으면 while문에서 빠져나가게 됩니다. 바로 코드를 통해 살펴보겠습니다.

우선 a라는 변수를 3으로 정해볼게요. a에서 1씩 줄어드는 숫자를 출력하는 while문을 만들겠습니다. a가 0보다 클 때만 while문이 돌아가도록 조건을 걸겠습니다.

| a = 3  while a > 0:  print(a)  a = a-1 |
| --- |

3

2

1

3부터 1까지 출력이 되면, a값은 0이 되므로 a>0 조건을 만족하지 못해 while문이 종료됩니다.

의도치 않게 무한으로 반복되는 while문을 만드는 경우가 빈번하니 작성에 주의하기 바랍니다.

## 2.4 조건문 : if문

if문은 특정 조건을 만족할 때 특정 코드를 실행하는 데 사용합니다. 예를 들어 홀수/짝수 여부를 판단해서 even, odd를 출력하게 할 수 있습니다. 코드로 확인해보겠습니다.

| x = 3 # ❶ 테스트할 값  if x % 2 == 0: # ❷ 기본 조건 : even 출력  print('even') else: # ❸ 나머지 : odd 출력  print('odd') |
| --- |

<출력 결과/>

odd

</>

❶ 테스트할 값 x를 3으로 지정해놓았습니다. ❷ 짝수일 때 even을 출력하는 코드입니다. x에 나눗셈에서 나머지를 구하는 % 연산을 수행합니다. ==가 이렇게 두 번 쓰이면 **같다/아니다**를 판별하는 연산자입니다. 결과적으로 2로 % 연산한 나머지가 0과 같으면 짝수이고, 1이면 홀수인 겁니다. ❸ 홀수일 때 odd를 출력합니다. 앞의 조건에 들지 않을 때 수행할 코드를 지정하는 데 else를 사용합니다.

조건식에 사용할 수 있는 비교 연산자는 다음과 같습니다.

▽ 파이썬 비교 연산자

| **연산자** | **설명** | **예시** |
| --- | --- | --- |
| == | 값이 같음 | (a == b) |
| != | 값이 같지 않음 | (a != b) |
| > | 왼쪽 값이 오른쪽 값보다 큼 | (a > b) |
| < | 왼쪽 값이 오른쪽 값보다 작음 | (a < b) |
| >= | 왼쪽 값이 오른쪽 값보다 크거나 같음 | (a >= b) |
| <= | 왼쪽 값이 오른쪽 값보다 작거나 같음 | (a <= b) |

이번에는 for문과 결합해 if문을 사용해보겠습니다. for문에 사용할 대상 리스트를 먼저 정하겠습니다.

| sample\_list = [-3,5,0,-1,2] |
| --- |

양수면 positive, 음수면 negative, 0이면 zero가 나오는 if문을 작성해봅시다. 기존에는 if와 else만 사용했는데, 이번에는 elif까지 사용합니다. elif는 원하는 만큼 계속 추가할 수 있습니다.

| for i in sample\_list:  if i > 0: # 기본 조건: i가 0보다 크면 positive를 출력  print('positive')   elif i < 0: # 추가 조건: i가 0보다 작으면 negative를 출력  print('negative')  else: # 나머지: 위의 두 경우 모두 해당하지 않으면 zero를 출력  print('zero') |
| --- |

negative

positive

zero

negative

positive

코드 해석은 어렵지 않을 겁니다.

파이썬에서는 코드 들여쓰기에 유의해야 합니다. 콜론 아래부터 들여쓰기로 들어간 부분은 코드가 해당 블록에 귀속된다는 의미입니다. for보다 if와 elif는 한 단계 더 들여쓰기되어 있습니다. for문에 귀속되는 구문이라는 뜻입니다. if/elif/else 바로 아래에 있는 print문은 한 단계 더 들여쓰기되어 있습니다. if/elif/else 각각 print문 한 줄이 귀속되어 있네요.

들여쓰기가 잘못되면 원하는 결과를 내지 못합니다. 예를 들어 살펴보겠습니다.

| for i in sample\_list:   if i > 0: # 기본 조건: i가 0보다 크면 positive를 출력  print('positive')   elif i < 0: # 추가 조건: i가 0보다 작으면 negative를 출력  print('negative')  else: # 나머지: 위의 두 경우 모두 해당하지 않으면 zero를 출력  print('zero') |
| --- |

File "main.py", line 8

print('zero')

^

IndentationError: expected an indented block

마지막 코드인 print('zero')의 들여쓰기가 잘못되어 에러가 발생했습니다. 파이썬에서는 이처럼 들여쓰기가 중요합니다.

## 2.5 파이썬 내장 함수

함수는 특정 작업을 수행하는 이름이 붙은 코드 블록입니다. 예를 들어 print() 함수는 출력 작업을 수행하는 print라는 이름이 붙은 코드 블록입니다. 파이썬은 70개가 넘는 내장 함수를 제공합니다.[[2]](#footnote-1)

모든 함수를 알고 있을 필요는 없습니다. 여기에서는 이 책에서 자주 사용하는 몇 가지만 정리하고 넘어가겠습니다.

| **함수명** | **설명** | **예문** |
| --- | --- | --- |
| abs() | 절대값 | <코드/>  abs(-3)  </>  <출력 결과/>  3  </> |
| len() | 데이터의 길이 확인 | <코드/>  len([1,3,5,7,9])  </>  <출력 결과/>  5  </> |
| max() | 최댓값 호출 | <코드/>  max([1,3,5,7,9])  </>  <출력 결과/>  9  </> |
| min() | 최솟값 호출 | <코드/>  min([1,3,5,7,9])  </>  <출력 결과/>  1  </> |
| range() | 데이터의 범위 지정 | <코드/>  list(range(1,10))  </>  <출력 결과/>  [1, 2, 3, 4, 5, 6, 7, 8, 9]  </> |
| round() | 반올림 | <코드/>  round(3.8)  </>  <출력 결과/>  4  </> |
| sum() | 합 | <코드/>  sum([1,3,5])  </>  <출력 결과/>  9  </> |

## 2.6 나만의 함수 만들기 : def

def 키워드를 사용하면 나만의 함수(사용자 정의 함수)를 만들 수 있습니다. 원형은 다음과 같습니다.

| def 함수명(매개변수)  # 수행할 코드  return 반환값 # return 코드는 생략 가능 |
| --- |

숫자를 넣으면 10을 더한 결과를 출력하는 간단한 함수를 만들겠습니다.

| def plus\_ten(x): #plus\_ten은 함수 이름, x는 이 함수에 들어갈 매개변수  return x + 10 #함수가 x에 10을 더하는 연산 |
| --- |

이제 만든 함수를 사용하겠습니다. plus\_ten() 함수 안에 5을 넣어봅시다.

| plus\_ten(5) |
| --- |

15

예상한 대로 15를 얻게 되었습니다.

이번에는 매개변수 2개를 받아 더한 값을 반환하는 함수를 만들겠습니다.

| def sum(a,b): # sum이라는 이름의 함수로, 매개변수는 a와 b  return a+b #a와 b의 합을 구하는 연산 |
| --- |

만든 sum() 함수에 10과 5를 넣어봅시다.

| sum(10,5) |
| --- |

15

이번에도 예상된 결과를 얻게 되었습니다. 마지막으로 매개변수에 기본값을 설정하는 함수를 만들겠습니다. 이번에도 매개변수는 a와 b 두 개인데, b에 특정값을 넣지 않으면 5가 자동으로 적용되게 하겠습니다.

| def sum\_five(a,b=5): # b에 5라는 기본값이 설정됨  return a+b |
| --- |

여기에서 10을 넣으면 과연 15를 구할 수 있을까요?

| sum\_five(10) |
| --- |

15

b값으로 기본값 5가 적용된 결과입니다.

그럼 10과 7을 인수로 넣어주면 어떻게 될까요?

| sum\_five(10,7) |
| --- |

17

17이 나왔습니다. 이처럼 호출할 때 기본값이 정의된 매개변수에 인수를 주지 않으면 기본값이 반영되지만, 인수를 주면 입력된 인수가 적용됩니다.

## 학습 마무리

이번 장에서는 파이썬에 대한 아주 기초적인 내용을 살펴보았습니다. 기초적인 내용이지만 이 시점에 모두 숙지하기보다는, 앞으로 직접 사용해보면서 차차 익혀나가는 걸 추천합니다. 여기서 소개한 파이썬 문법은 극히 일부에 불과합니다. 머신러닝까지 이르는 허들을 낮추고자 의도적으로 간단히 파이썬 문법을 다뤘습니다. 처음부터 너무 심도 있게 파이썬을 공부하면 정작 힘을 쏟아야 할 때 기운이 빠집니다. 더 깊은 내용은 머신러닝 알고리즘을 다루면서 필요할 때마다 다루겠습니다. 파이썬 문법을 깊이 공부하실 분은 별도의 책을 선정해 공부하시기 바랍니다.

1. TIOBE Index for January 2022 [↑](#footnote-ref-0)
2. 파이썬 3.10.2 버전 기준. https://docs.python.org/3/library/functions.html [↑](#footnote-ref-1)