Python 기초 문법

파이썬의 기초 문법으로는 식별자, 변수, 자료형, 데이터 형변환, 연산자에 대해서 학습합니다.

1. Comment와 Print

실습에 들어가기 전에 코드에 대한 설명을 하는 comment와 값을 출력하는 print에 대해서 알아봅니다.

1.1 comment

코드에 대한 설명인 주석을 입력할때는 해당 줄의 가장 앞에 #을 넣어줍니다. 가장 앞 라인에 #이 들어가면 해당 라인은 코드로 인식되지 않아 실행되지 않습니다.

```
# comment
```

1.2 print

데이터를 출력할때는 print 함수를 사용합니다.

```
print(123)
```

파이썬에서 코드의 길이가 길어 여러줄을 사용해야 할때는 \ 를 사용합니다.

```
print(12345\
7890)
```

2. 식별자 (Identifiers)

변수, 함수, 클래스, 모듈 등을 구분하기 위해 사용되는 이름을 식별자라고 합니다. 이 식별자는 몇가지의 규칙이 있습니다.

- 대소문자를 구분
- 소문자(a~z), 대문자(A-Z), 숫자(0~9), 언더스코어(_) 사용 가능
- (_)를 제외한 특수문자는 사용 불가
- 가장 앞에 (__) 사용 지양 (reserved global variable)
- 가장 앞에 숫자 사용 불가
- 예약어 사용 불가
  - 예약어 : Fasle, 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

3. 변수 (Variable)

변수는 메모리에 데이터를 저장하기 위한 저장공간의 이름으로 변수에는 값을 가지게 됩니다.

3. 1 변수 선언

a 라는 이름의 변수를 만들고 변수에 1 이라는 숫자를 값(value)을 넣어줍니다.

```
a = 1
```

b 라는 이름의 변수를 만들고 “python” 이라는 문자열을 값으로 넣어줍니다.

```
b = “python”
```

3. 2 식별자의 규칙에 맞지 않는 변수 할당

```
# _ 를 제외한 특수문자 사용불가
abc% = 10

# 가장 앞에 숫자 사용 불가
1_abc = 10

# 예약어 사용 불가
class = 10
```

3. 3 다중할당

변수의 할당은 한줄에 여러개의 변수에 값을 할당할수 있으며 같은 값을 여러개의 변수에 할당할수 있습니다.

2. 3. 1 한줄에 여러개의 변수 할당

c 변수에 5라는 정수 값을 할당하고, d 변수에 “dss” 라는 문자열 값을 할당합니다,

```
c, d = 5, “dss”
```

2. 3. 2 한줄에 같은 값을 여러개의 변수에 할당

e, f, g 변수에 “datascience” 라는 값을 할당합니다.

```
e = f = g = “datascience”
```

4. 자료형 (Datatype)

파이썬에서의 자료형에는 크게 Number, Boolean, String 세가지가 있으며 세가지 자료에 대한 집합인 Collection으로는 List, Tuple, Dictionary, Set 네가지가 있습니다.

4. 1 Numbers

정수인 integer와 실수인 float, 복소수인 complex 세가지로 구분이 됩니다.

4. 1. 1 int

정수형 데이터로 python 2.x 버전에서는 (-2**31 - 1 ~ 2**31) 의 숫자를 표현할수 있고, python 3.x 에서는 거의 무한대로 사용이 가능합니다.

```
a, b, c, d = 0, 998374, -188723, +29993
print(type(a),type(b),type(c),type(d))
print(a, b, c, d)
```

4. 1. 2 float

소수점이 있는 실수를 나타냅니다.

```
a, b, c, d = 0.0, 998.3223, -188723.1233, +29993.111
print(type(a),type(b),type(c),type(d))
print(a, b, c, d)
```

4. 1. 3 complex

복소수로 실수부와 허수부, 켤레복소수, 복소수의 크기, 복소수의 연산등을 수행할수 있습니다.

```
a = 2 + 3j
b = complex(3, -4)
print(a, b)

# 실수부, 허수부
print(a.real, a.imag, b.real, b.imag)

# 복소수 연산
print(a + b, a - b, a * b, a / b)

# 켤레복소수
print(a.conjugate(), b.conjugate())

# 복소수의 크기
print(abs(a), abs(b))
```

4. 2 Boolean

True와 False 두가지로 이루어진 데이터 타입입니다.

```
a, b = True, False
print(type(a), type(b))
print(a, b)
```

4. 3 String

4. 3. 1 문자열 선언

문자들의 집합인 문자열을 나타내는 데이터 타입입니다. “” 나 ‘’ 안에 문자열을 넣어 변수를 선언 합니다.

```
a, b, c, d = "FastCampus", "데이터사이언스", '데이터"Science', "데이터'Science"
print(type(a), type(b), type(c), type(d))
a, b, c, d
```

여러줄을 작성할때는 “”” 문자열 “”” 이나 ‘’’ 문자열 ‘’’을 사용합니다.

```
ms = """FastCampus
데이터사이언스
Science"""
ms
```

연산자를 이용하여 문자열을 반복하여 저장 할수 있습니다.

```
a = "Fast " * 4
a
```

4. 3. 2 문자열 offset

오프셋을 사용하여 문자열을 자르거나 특정 위치의 문자를 출력할수 있습니다. 오프셋은 [start:end:stride] 형태로 사용 합니다.

오프셋에 하나의 정수만 넣으면 정수에 위치한 문자가 출력됩니다. 양수는 왼쪽에서 오른쪽으로 0부터 시작되고 음수는 가장 뒤에서부터 시작합니다.

```
a = "abcdefghijk"
a[0], a[5], a[-1], a[-5]
```

오프셋으로 문자열을 슬라이싱 할 수 있습니다. start에 아무것도 안쓰면 가장 앞을 의미하며 end 부분에 아무것도 안쓰면 가장 마지막 index 값을 의미합니다. stride에 아무것도 안쓰면 1이 들어간것과 같습니다.

```
a = "abcdefghijk"
a[1:3], a[:5], a[-3:], a[::2]
```

stride에 -1을 넣어 문자열을 거꾸로 만들수도 있습니다.

```
a = "abcdefghijk"
a[::-1]
```

len() 함수를 이용하여 문자열의 길이를 알수 있습니다.

```
a = "abcdefghijk"
len(a)
```

end에 아무것도 안쓴것과 len(a) 는 같습니다. 아래의 코드는 같은 문자열을 출력합니다.

```
a = “abcdefghijk"
a[:len(a)], a[:]
```

4. 3. 3 String Method

문자열 함수를 이용하여 문자열을 원하는 대로 수정할수 있습니다. 문자열 함수에 대해서는 (https://docs.python.org/3/library/stdtypes.html#string-methods) 를 참고 하시면 됩니다.

주요 문자열 메서드

- upper() : 대문자로 변환
- lower() : 소문자로 변환
- find() : 해당 문자열 위치 리턴 (없으면 -1 리턴)
- index() : 해당 문자열 위치 리턴 (없으면 value error 리턴)
- count() : 문자열 갯수 리턴
- lstrip() : 왼쪽 공백 제거
- rstrip() : 오른쪽 공백 제거
- strip() : 양쪽 공백 제거
- replace() : 특정 문자열 치환
- split() : 특정 문자열로 분리하여 리스트 형태로 리턴
- endswith() : 마지막 문자열이 일치하면 True 리턴
- join() : 문자열 리스트를 결합

```
a = "  FAST campus datascience  "

print(a.upper())
print(a.lower())

print(a.find("c"))
print(a.find("q"))

print(a.index("c"))

print(a.count(“a"))

print(a.lstrip())

print(a.rstrip())

print(a.strip())

print(a.replace("a","b"))

print(a.split(" "))

print(a.rstrip().endswith("datascience"))

a = ["fast", "campus", "data", "science"]
print(“-“.join(a))
```

4. 4 List

순서가 있는 데이터들의 집합을 가지는 데이터 타입입니다. List, Tuple, Dictionary, Set 의 데이터 타입을 Collection이라고 합니다.

4. 4. 1 List 선언

[ ] 의 기호를 사용하여 리스트를 선언할수 있습니다. 리스트의 데이터로 리스트를 넣을수도 있습니다.

```
a = ["fast", "campus", "data", "science"]
b = [1, 2, [3, 4]]
c = [1, "fast", True]
print(type(a), type(b), type(c))
a, b, c
```

4. 4. 2 List offset

리스트도 문자열과 같이 오프셋을 사용할수 있습니다.

```
a = ["fast", "campus", "data", "science"]
print(a[1], a[-1], a[1:3], a[:], a[::-2])
a[-2] = "pdj"
a
```

4. 4. 3 List Method

리스트 함수를 이용해 리스트 데이터를 가공할수 있습니다.
주요 리스트 메서드

- append() : 가장 마지막에 데이터 추가
- sort() : 오름차순 정렬
- reverse() : 내림차순 정렬
- index() : 데이터 위치 리턴
- insert() : 특정 위치에 데이터 추가
- remove() : 해당되는 데이터 값을 삭제 (오프셋이 가장 작은 데이터 하나만 삭제)
- pop() : 가장 마지막 값을 리턴하고 마지막 값을 삭제

```
a = ["fast", "campus", "data", "science"]

a.append("fighting!")
print(a)

b.sort()
print(b)

b.reverse()
print(b)

print(a.index('campus'))

a.insert(2, 'index')
a.insert(5, 'index')
print(a)

a.remove('index')
print(a)

print(a.pop())
print(a)
```

4. 4. 4 List 데이터 수정

오프셋을 이용하여 리스트의 데이터를 수정할 수 있습니다.

```
a = [1,2,3,4,5]

# offset을 이용하여 데이터 수정
a[1] = "Fast"
print(a)

a[1:4] = "a"
a
```

4. 4. 5 List Copy

리스트를 복사하기위해서는 copy() 함수를 사용해야 합니다. b = a 형태로 복사를 하게 되면 같은 저장공간에 두개의 이름을 가지게 되어 a를 변경하게 되면 b도 변경이 됩니다. 별도의 저장공간을 가지게 하려면 copy() 함수를 사용해야 합니다.

```
a = [1,2,3]
b = a
print(a, b)

# a를 바꿨는데 b의 데이터도 같이 바뀜
a[2] = 4
print(a, b)

c = a.copy()
print(a,c)

# a를 바꿔도 c의 데이터는 바뀌지 않음
a[2] = 5
print(a,c)
```

4. 5 Tuple

List와 같이 순서가 있는 데이터 타입이지만 데이터를 변경할수 없는 특징이 있습니다. Tuple을 사용하는 이유는 Tuple이 List보다 컴퓨터의 자원(메모리)을 적게 사용합니다. 또한 아래에서 배우는 Dictionary의 key로 사용 될수 있습니다.

4. 5. 1 Tuple 선언

튜플은 , 로 구분하여 선언할수 있고 , 로 구분하고 괄호로 묶어서 선언할수 있습니다. 튜플에서도 문자열이나 리스트에서와 같이 오프셋을 사용하여 데이터를 가공할수 있습니다.

```
a = 1, 2, 3, 4
b = "fast", "campus", "data", "science"
c = 1, "fast", True
d = (1, "fast", True)

print(type(a), type(b), type(c), type(d))
print(a, b, c, d)
a[2], a[1:3]
```

튜플의 데이터는 수정이 불가능 합니다. 수정하려고 하면 “TypeError: 'tuple' object does not support item assignment” 와 같은 에러가 발생합니다.

```
a = 1, 2, 3, 4
a[1] = 3
```

같은 데이터를 변수에 저장하면 리스트 보다 튜플이 저장공간을 덜 사용합니다. 그러므로 변경이 되지 않는 순서가 있는 데이터의 집합의 경우 리스트보다 튜플을 사용하는것이 컴퓨터의 자원을 사용하는데 더 효율적입니다.

```
import sys

# getsizeof의 단위  : byte
l = [1,2,3,4,5]
print(sys.getsizeof(l))
t = tuple(l)
print(sys.getsizeof(t))
```

4. 6 Dictionary

딕셔너리는 데이터의 순서가 없고 key와 value의 쌍으로 데이터가 모여있는 데이터 타입니다.

4. 6. 1 Dictionary 선언

딕셔너리는 {} 기호 안에 키 : 값 형태로 선언합니다({ 키:값 }). 키값에는 정수나 문자열의 데이터 타입이 사용 가능합니다.

```
dic = {
    1:"one",
    "A": ["data", "science"],
    "숫자": 1234,
    }

type(dic), dic
```

4. 6. 2 Dictionary 데이터 수정

딕셔너리는 키값으로 데이터를 수정할 수 있습니다.

```
dic[1] = "하나"
dic["A"] = "알파벳"
dic
```

순서가 없는 데이터 타입이므로 오프셋의 슬라이스는 사용이 불가능합니다. 아래의 코드는 에러가 발생합니다.

```
dic[1:3]
```

4. 6. 3 Dictionary Method

딕셔너리 함수를 이용해 리스트 데이터를 가공할수 있습니다.

주요 딕셔너리 메서드

- keys() : 키를 리턴
- values() : 값을 리턴
- items() : 키와 값을 리턴
- clear() : dictionary 데이터를 모두 삭제
- get() : 매개변수에 해당하는 값을 리턴
- copy() : 리스트와 마찬가지로 다른 저장공간을 가지는 데이터를 대입
- update() : 데이터를 업데이트

```
print(dic.keys())
print(dic.values())
print(dic.items()) # for 문에서 key value로 루프를 돌릴때 사용됩니다.

dic.clear()
print(dic)

dic = { 1:”one", 2:”two", 3:”three" }
print(dic.get(2))
print(dic.get(4))
# print(dic[4])
print(dic.get(4,"no_data")) # get을 사용하면 error를 예방할수 있다.

dic2 = dic
dic[1] = "하나"
dic3 = dic.copy()
dic[1] = "A"

a = {1:"a", 2:"b"}
b = {2:"c", 3:"d"}
print(a)
a.update(b)
a
```

4. 7 Set

셋은 중복되는 데이터가 없는 집합 형태의 데이터 타입입니다. 교집합, 합집합, 차집합과 같은 집합의 연산도 가능하며, 리스트 데이터에서 중복을 제거할때 사용하기도 합니다. 또한 순서가 있는 데이터 타입이 아니기 때문에 오프셋을 이용한 특정 인덱스 값을 가져오거나 슬라이싱으로 데이터 수정이 불가능합니다. 수정을 위해서는 리스트로 데이터 형변환후 데이터를 수정하고 다시 셋 데이터 타입으로 바꿔 주어야 합니다.

4. 7. 1 Set 선언

셋은 리스트 형태의 데이터에 set() 으로 형변환를 해주는 방법으로 선언을 하며 중복된 데이터는 제거 됩니다.

```
s = set([1,2,3,4,5,1,2,3])
type(s), s
```

4. 7. 2 집합의 연산

```
s = set([1,2,3,4,5,1,2,3])
type(s), s
```

4. 7. 3 Set 데이터 수정

```
s = list(s)
s[4] = 10
s = set(s)
s
```

5. 데이터 형변환

서로 다른 데이터 타입사이에 데이터의 형변환이 가능합니다. 하지만 데이터 타입에 따라서 형변환이 되는 데이터 타입이 있고 되지 않는 데이터 타입이 있습니다.

5. 1 문자열과 숫자

문자열과 숫자 사이의 형변환이 가능합니다.

```
s = "1234"
n = int(s)
print(type(n), n)

s = "1234"
n = int(s)
print(type(n), n)
```

하지만 숫자문자가 아닌 문자열을 숫자로 형변환하는것은 불가능합니다.

```
s = "a"
n = int(s)
print(type(n), n)
```

5. 2 boolean 형변환

문자열이나 리스트나 정수의 데이터가 boolean으로 어떻게 형변환 되는지 살펴보겠습니다. 숫자는 0이 False, 문자열은 “” 이 False, 리스트에서는 [ ] 가 False로 형변환 됩니다.

```
print(bool(""), bool("asedf"))
print(bool(-1), bool(0), bool(1), bool(0.0), bool(0.1))
a, b = [ ], [1,2]
print(bool(a), bool(b))
```

이 내용은 데이터 존재의 유무를 판별할수 있어서 뒤에 배우는 조건문에서 조건을 정의할때 알아두면 더 깔끔한 코딩을 할수 있습니다.

```
a, b = [ ], [1,2,3]

if len(a) > 0:
    print(a)
else:
    print("no_data")

if b:
    print(b)
else:
    print("no_data")
```

5. 3 문자열의 리스트 형변환

문자열을 리스트로 형변환하면 문자 하나씩 순서대로 리스트의 데이터로 추가되는 형태로 형변환 됩니다.

```
s = "fast"
l = list(s)
l
```

5. 4 튜플과 딕셔너리 형변환

튜플과 딕셔너리 사이에는 형변환이 가능합니다.

5. 4. 1 튜플의 딕셔너리 형변환

튜플의 (key, value) 형태의 나열로 되어 있는 튜블의 데이터는 딕셔너리로 형변환이 가능합니다.

```
t = ((1,"one"),(2,"two"))
dict(t)
```

5. 4. 2 딕셔너리의 튜플 형변환

딕셔너리를 튜플로 형변환 하면 딕셔너리의 키 데이터만 형변환 됩니다. key와 value 데이터를 모두 사용하여 형변환 하려면 items() 함수를 사용합니다.

```
d = {1:"one", 2:"two"}
t = tuple(d)
print(t)
t = tuple(d.items())
print(t)
```

6. 연산자 (Operators)

6. 1 산술 연산자 (Arithmetic Operators) : `+, -, *, /, //, %, **`

```
print(2 + 4)
print(2 - 4)
print(2 * 4)
print(10 / 3)
print(10 // 3)
print(10 % 3)
print(3 ** 4)
```

6. 2 비교 연산자 (Comparison Operators) : `==, !=, >, <, =>, =<`

데이터를 비교할때 사용하며 결과가 True와 False로 나타납니다.

```
a, b = 10, 20

print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a>=b)
print(a<=b)
```

6. 3 할당 연산자 (Assignment Operators) : `=, +=, -=, *=, /=, %=, //=, **=`

```
a = 5
print(a)

a += 10
print(a)

a -= 5
print(a)

a *= 10
print(a)

a /= 5
print(a)

a %= 8
print(a)

a //= 2
print(a)

a **= 3
print(a)
```

6. 4 비트 연산자 (Bitwise Operators) : `&, |, ^, ~, <<, >>`

```
a = 3 # 011
b = 5 # 101

print(a&b, bin(a&b))
print(a|b, bin(a|b))
print(a^b, bin(a^b))
print(~a, bin(~a))
print(~b, bin(~b))
print(a<<2, bin(a<<2))
print(b>>1, bin(b>>1))
```

6. 5 논리 연산자 (Logical Operators) : `and, or, not`

```
print( True and True )
print( True and False )

print( True or True )
print( True or False )

print( not(True and True) )
print( not(True and False) )
```

6. 6 멤버 연산자 (Membership Operators) : `in, not in`

```
a = [1,2,3,4]
print(1 in a)
print(5 in a)
print(1 not in a)
```

6. 7 식별 연산자 (Identity Operators) : `is, is not`

숫자나 문자열의 경우는 값을 비교합니다.

```
a = "a"
b = a
print( a is b )

b = "a"
print( a is b )

print( a is not b )
```

컬렉션 데이터의 경우에는 주소값을 비교 합니다.

```
a = [1,2,3]
b = a
print( a is b )

# 데이터가 저장되어 있는 주소가 같은지 확인합니다.
a = [1,2,3]
b = a.copy()
print( a is b )

print( a is not b )
```