자료형(data type)

변수의 자료형, 데이터의 종류에 대해서 살펴 보겠습니다. 파이썬에서 사용하는 자료형은 보통 숫자형, 문자(열)형, 논리형이 자주 사용되는 편이고, 리스트, 튜플, 딕셔너리, 셋등의 자료형이 있습니다.

먼저, 이번시간에는 숫자형 자료형에 대해서 살펴 보도록 하겠습니다. 그 전에, 기본적인 규칙에 대해서 알려드리겠습니다.

```
# 이것은 파이썬의 한줄 주석입니다
# 편집기에서 ctrl + / 주석 토글(적용,해제)
'''
이것은 여러줄을 주석으로 처리할때 사용합니다.
쌍따옴표를 사용하기도 합니다.
'''
# 문장의 끝에 세미콜론(;)은 생략 가능합니다.
# 공백은 사용하지 않도록 주의하세요
```

프로그래밍에서는 '자료'를 처리해주는 역할을 합니다. 프로그래밍에서 처리할 수 있는 모든 것을 자료라 하는데, 파이썬 역시 다양한 자료를 다루고 처리해주는 역할을 합니다. 개발자들이 쉽게 사용할 수 있도록 기능과 역할에 따라 구분한 '자료'를 자료형(data type)이라고 합니다. 기본적인 자료형인 숫자형, 문자열형, 논리형에 대해서 살펴봅시다.

▼ 숫자형

- 정수형(int)
  ex> 2, 5, 10, 100
- 실수형(float)
  ex> 2.0
- 복소수형(complex) - 공학 분야에서 주로 사용
  ex> 어떤 수의 제곱 = -1 (허수)

```
'''
  1.변수는 데이터를 담는 그릇, 컨테이너 입니다.
  2.변수의 이름은 예약어(keyword)를 사용할 수 없습니다.
  3.변수의 이름은 의미를 부여하여 작성하도록 합니다.
  4.변수의 이름은 문자 또는 _의 기호를 사용하도록 합니다.
```

```
  5.공백은 사용할 수 없습니다.
  6.일반적으로 Camel Case 또는 snake_case를 사용하여 네이밍 합니다.
'''
# 간단한 숫자형 변수 선언 - 의미가 없음
a = 10
b, c = 20, 30
d = 2.0

# 변수의 메모리상 위치 확인하기 - id() 함수
print(id(b))
print(id(c))

# camel case 사용 - guest + name
guestName = '홍길동'

# snake case 사용 - gender + type
gender_type = 'male'

# 출력문 print() 사용해보기
print(a+10)

# 실수형 변수 사용
discount_rate = 0.15
price = 1000000
final_price = price - (price * discount_rate)

# 실수형 데이터를 정수형으로 변환하기 - int() 함수
print(int(d))

# 정수형 데이터를 실수형으로 변환하기 - float() 함수
# 숫자형 데이터를 문자형으로 변환하기 - str() 함수

# 자료형 타입 확인하기
print(type(a))
print(type(guestName))

# 간단한 계산해보기 : +, -, *, /, %, **, //
num1 = 10
num2 = 3
print(num1 + num2)
print(num1 - num2)
print(num1 * num2)
print(num1 / num2)
print(num1 % num2)
print(num1 ** num2)
print(num1 // num2)
```

▼ 문자(열)형

글자들이 길게 나열된 것을 문자(열)형 이라고 합니다. 기본적으로 쌍따옴표(")나 외따옴표(')로 묵여있는 모든것을 문자(열)형 이라고 생각하면 됩니다.

```
# 문자열 출력하기
print('Hello World')
```

```
print("hello", "world")
print("hello"+" python "+"world")
print("=" * 10)
print(5 * '-')

# 문자열 출력시 구분하기 - sep옵션 , end 옵션
print("hello","world","python",sep="/")
print("hello", end="")
print("world")

# 문자열 내에 "나 '를 입력할때 주의할 것
print("I'm fine, thanks you!")

# 이스케이프(escape) 문자 사용 : \
print("I\"m fine, thanks you!")

# 문자열 출력시 포맷팅(formatting) 사용해보기
# f-string 방식은 파이썬 3.6 이상 사용가능
print('최초 구입 가격은 %s원 입니다' % format(price, ','))
print('할인율은 %d%% 입니다' % (discount_rate * 100))
print(f'최종 구입 가격은 {final_price : ,}원 입니다')

# 문자열 인덱싱 [인덱스]
python_string = "life is too short, you need python"
print(python_string[2])
print(python_string[-6])
# mutable vs immutable - 값의 변경 vs 값의 불변
# 숫자형, 문자열형, 튜플은 immutable, 리스트와 딕셔너리는 mutable
# 아래 코드는 실행시 에러가 발생합니다.
# python_string[0] = 'w'

# 문자열 슬라이싱1 [시작:끝]
print(python_string[0:4])

# 문자열 슬라이싱2 [시작:끝:단계]
print(python_string[0:4:2])
```

▼ 논리형

논리형 데이터는 True 또는 False를 가질 수 있습니다. 명시적으로 True, False를 표시하거나 연산자를 이용해 논리형 데이터를 가지게 됩니다.

```
# 명시적으로 True, False 지정하기
isTrue = True
isFalse = False
true_or_false = 4 > 5

print(isTrue)
print(10 == '10')
print(20 != '20')
```

```
print(10 > 10)
print(true_or_false)
print(5<=5)
print(5<10)


# 제어문 - 조건문 - 단순if문
if isTrue:
    print("true 입니다")

if isFalse:
    print("false 입니다")

if true_or_false:
    print("이 문장이 출력되면 true 입니다")

# 논리 연산자 and - 하나라도 false면 전체가 false
result1 = 7 < 10 and 7 > 5
if result1:
    print('참')
else:
    print('거짓')

# 논리연산자 or - 하나라도 true면 전체가 true
result2 = 5 <7 or 10 >= 5
if result2:
    print('참')
else:
    print('거짓')

# 논리연산자 not - 참, 거짓을 서로 바꾸어준다
print(not isFalse)

# 홀수, 짝수 판별하기
# 파이썬은 block scope 대신 들여쓰기를 합니다. 주의!!
like_num = int(input("1~10중 숫자를 하나 입력하세요"))
if like_num % 2 == 0:
    print(f"당신이 좋아하는 숫자 {like_num}은(는) 짝수입니다")
else:
    print(f"당신이 좋아하는 숫자 {like_num}은(는) 홀수입니다")
```

▼ 그 밖의 자료형

- 리스트(list)

```
# 변수 - 하나의 데이터 값을 저장, 변하는 자료형
# 리스트 선언 - sequence type, 순번(index)를 통해 값을 찾을 수 있음
# 해시(hash), 연관배열(associative array)
list_a = [1, 32, 100, 'python', True, False, ['AA', 'BB']]

# 비어있는 리스트
list_b = []
# list_b = list()
```

```
# list()
# list_b = list((1,2,4,9))

# 리스트 인덱싱, 슬라이싱 - 문자열의 인덱싱, 슬라이싱과 같음
print(list_a[1+1])
print(list_a[1-1])
print(list_a[1*1])
# / : 실수(float), error!
print(list_a[4/2])
print(list_a[3])
print(list_a[4:6])
print(list_a[-1][1])

# 리스트 함수
list_b = [2, 4, 8, 10, 100, 8, 2]
print(max(list_b))
print(min(list_b))
print(list_b.count(2))
print(list_b.index(4))
print('리스트의 길이 : ',len(list_b))

# 리스트 연산
print(list_a[2] + list_b[5])
print(list_b * 2)
print(list_a[-1][0]+str(list_b[0]))

# 리스트의 수정
list_c = [1,2,3,4,5]
list_c[0] = 2
print(list_c)

# 리스트에 값 추가
list_c.append(99)
list_c.insert(0,100)
print(list_c)
list_c.extend([True,False,777])


# 리스트의 값 삭제 - del, remove, pop
list_c.pop()
list_c.pop(3)
del list_c[0]
list_c.remove(3)
print(list_c)
list_c.clear()

# 리스트의 값 정렬 - .sort(), .reverse()
list_c.reverse()
list_c.sort()
list_c.shuffle()


# 그밖에.. - .index(), .count()
print(list_c.count(1))
print(list_c.index(4))

# 리스트와 for 반복분
```

```
list_d = [1, 2, 4, 7, 9]
for num in list_d:
    print(num)

list_empty = []

for number in list_d:
    if number % 2 == 1:
        list_empty.append(number * 3)

# 리스트 내포(list comprehension) 사용시 - 홀수 원소에 3을 곱해서 result 에 추가
# 리스트 변수 = [ (변수를 활용할 값) for (변수명) in (순회할 수 있는 값) (조건)]
# list_e = [num for num in range(10)]
list_e = [number * 3 for number in range(10)]
print(list_e)

list_e = [number * 3 for number in list_d if number % 2 == 1]
list_f = [num ** 2 for num in list_d]
print(list_e)
print(list_f)

# 리스트 비우기(=제거) - .clear()

list_g = [0, 1, 3, 9, 15, 33]
list_g.clear()
print(list_g)

# 리스트 - 멤버쉽 테스트(in, not in)

print(33 in list_g)
print(99 in list_g)

# 랜덤 모듈을 이용한 난수 발생, 리스트에 추가하기
import random
list_nansu = []
for value in range(1,10):
    nansu = random.randint(1,10)
    list_nansu.append(nansu)
print(list_nansu)

# sum() - 리스트의 요소의 합을 구하는 함수
print(sum(list_d))

# Quiz1. 중복되지 않는 난수를 생성해서 list_nansu에 추가하고 총 길이값을 출력해보세요
# Quiz2. list_nansu의 숫자 총합을 출력해보세요
'''
import random
list_nansu = []

for value in range(1,10):
    nansu = random.randint(1,10)
    if nansu in list_nansu:
        nansu = random.randint(1,10)
    else:
        list_nansu.append(nansu)
print(list_nansu)
print('리스트 길이 : ',len(list_nansu))
nansu_sum = 0
```

```
for su in list_nansu:
    nansu_sum += su
print('리스트 총합 : ',nansu_sum)
'''

# Quiz3. 1~100까지 숫자를 생성하고 이것을 2진수로 변환, 0의 갯수가 1개인 숫자의 합을 출력
'''
total = 0
for i in range(1,101):
    if f'{i:b}'.count('0') == 1:
        print(f'{i:b} - {i}')
        total += i
print(total)
또는
output = [i for i in range(1,101) if f'{i:b}'.count('0') == 1]
print(sum(output))
'''

# 리스트가 중복되면 반복문을 두번 사용
```

- 튜플(tuple)

```
# 리스트와 튜플은 유사하나, 다른점이 있다. sequence type!
# [] vs ()
# mutable vs immutable : 한번 생성된 객체는 수정이 불가능함


# 튜플의 선언 vs 리스트의 선언
list_a = [1, 32, 100, 'python', True, False, ['AA', 'BB']]
tuple_a = (10,'AA',True, False,(99,'python'))
print(tuple_a, type(tuple_a))

# (는 생략할 수 있지만, ,는 포함시킬것
tuple_b = 99, 88, 77, 66
print(tuple_b, type(tuple_b))

# tuple_c = (99,)
# tuple_d = 100,
# tuple_e = ()
# tuple_f = tuple()

# 리스트는 값을 수정할수 있지만,
list_a[2] = 999

# 튜플은 수정할 수 없다. 에러발생!

# tuple_a[1] = 100
# del tuple_a[-1]

#튜플 멤버쉽 테스트 : in, not in
tuple_b = 1, 2, 3, 'A', '4', 6.5
for i in tuple_b:
    print(i)
print(1 in tuple_b)
print('B' not in tuple_b)
```

```
# 튜플의 인덱싱
print(tuple_a[0], tuple_a[-1][0])

# 튜플의 슬라이싱
print(tuple_a[0:2])
print(tuple_a[3:])

# 튜플의 연산
print(tuple_a + tuple_b)
print(tuple_a * 3)

# 튜플의 길이
print('튜플 길이 : ',len(tuple_b))

# 튜플 내포(tuple comprehension) - 값의 수정이 없다면
tuple_d = (i for i in range(10))
print(d)
# <generator object <genexpr> at 0x000002D6C4407C10> : generator (메모리)
# for 반복문을 이용해 generator에 접근할 수 있음
# generator vs list : 소비적인 객체 vs 재사용 객체
```

- 딕셔너리(dictionary)

```
# 키:값의 대응관계를 나타내는 딕셔너리, 리스트는 index, 딕셔너리는 key!
# dictionary : 사진, {'용어':'뜻',...} 형태
# 딕셔너리의 key는 유니크하지만(변경x) value는 무관-str, int, tuple, list, etc

# 딕셔너리의 선언
dict_a = {
'name':'python',
'version':3.9,
'status':'stable',
'issue':None
}
# 빈 딕셔너리 생성, 키/값 추가
dict_b = {}
dict_b['name'] = 'i-clicker'
dict_b['model'] = 'GM-110'
dict_b['price'] = 15000

# 그 외의 방법
'''
dict1 = dict(lev=100, city='dark')
dict2 = dict(zip(['type','blood'],['knight','O']))
dict3 = dict([('hp',100),('mp',1200),('kill',100), ('death',10)])
dict4 = dict({'id':'nunununa', 'master':'magic', 'level':99})
'''
# 키 값에 접근하기
print(dict_b['name'])
print(dict_b['price'])
#print(dict_b.get('name'))

# 같은 키값이 있을경우, 기존의 값을 업데이트
dict_a['url'] = 'www.python.org'
```

```
dict_a['module'] = 'default'
dict_a['issue'] = 'update'
print(dict_a)

# 존재하지 않는 키값에 접근시 KeyError 에러발생
# 딕셔너리 값 얻기 - 인덱싱x, 슬라이싱x
# 선언은 {,}로 하지만, 접근은 [,]를 이용해 키(key)로 접근
print(dict_a['version'])
print(dict_a['author'])
del dict_a['color']

# 추가) 존재하기 않는 키에 접근할려고 할때 .get() 함수 사용-> error가 아닌 None을 반환
print(dict.get('awesome'))

# 딕셔너리 키 제거 --> 키,값이 쌍으로 제거
del dict_a['status']
print(dict_a)

# .clear() - 딕셔너리의 모든 키/값 제거
dict_b.clear()
print(dict_b)

# 딕셔너리 멤버쉽 테스트
print('price' in dict_b)
print('color' not in dict_b)

# 딕셔너리 키의 갯수(길이) 구하기 - 키:값은 1:1관계
print(len(dict_b))

'''
dict_b = {}
dict_b['name'] = 'i-clicker'
dict_b['model'] = 'GM-110'
dict_b['price'] = 15000

while True:
    key_name = input('> 접근하고자 하는 키(A-키 추가, D-키 삭제, Q-종료) : ')
    if key_name.upper() == 'A':
        add_key = input('+ 추가하고자 하는 키 : ')
        add_value = input('+ 추가하고자 하는 값 : ')
        dict_b[add_key] = add_value
        print('새로운 키를 추가했습니다')
        continue
    elif key_name.upper() == 'Q':
        print('사용자가 종료하였습니다')
        break
    elif key_name == '':
        print('접근하고자 하는 키를 입력하세요')
        continue
    elif key_name.upper() == 'D':
        del_key = input('- 삭제하고자 하는 키 : ')
        del dict_b[del_key]
        print(del_key)
    else:
        if key_name not in dict_b:
            print('존재하지 않는 키에 접근합니다.')
            continue
        else:
```

```
            print(dict_b[key_name])
            continue
'''

# 딕셔너리 함수 - keys(), values() / list반환 vs .items() / tuple 반환
# type()으로 확인해보기

dict_keys = dict_a.keys()
dict_values = dict_a.values()
print(dict_keys, type(dict_keys))
print(dict_values, type(dict_values))
print(dict_a.items())
'''
dict = {
    'name': 'new name',
    'type': 'new type',
    'indigredient': ['mango', 'banana', 'nitro'],
    'origin': 'korea'
}
print(dict.items())
for key, value in dict.items():
    print(key,':',value)
'''

# 리스트 내포(list comprehension) -> 딕셔너리(set comprehension) 내포
dict_some = { x for x in range(10)}
print(dict_some, type(dict_some))

odd = { i for i in range(10) if i % 2 == 1}
even = { j for j in range(10) if j % 2 == 0}

# .get() vs dict['some_key'] - None vs 에러
```

- 셋(set)

```
# 집합을 쉽게 처리하기 위한 자료형
# 딕셔너리와 같은 {,}를 이용해 생성하고 값은 ,(comma)로 구분
set_a = {}
print(type(set_a))
# <class 'dict'>

# 빈 셋트생성
set_b = set()
print(type(set_b))
# <class 'set'>

set_c = {1, 2, 3, 4, 1, 2, 3}
print(set_c)
# {1, 2, 3, 4}

# 문자열에서 문자를 몇개 사용했는지 파악한다면? str to set!
str_a = '안녕안녕안녕하세요'
print(set(str_a))
# {'녕', '요', '안', '세', '하'}
```

```
# 1. 중복을 허용하지 않음
# 2. 순서가 없음 (=순서가 보장되지 않음)
set_a = set(['hello','python'])
set_b = set('hello')
set_c = set('python')
set_d = set([1,3,5,7,9])
set_e = set([0,2,3,4,5])

# print(set_a, type(set_a))
# print(set_b)

# 교집합
print(set_d & set_e)
print(set_d.intersection(set_e))

# 합집합
print(set_d | set_e)
print(set_d.union(set_e))

# 차집합
print(set_d - set_e)
print(set_d.difference(set_e))

# 부분집합
print(set_a.issubset(set_b))
# True, False

# 값 추가하기 : .add()
# 값 제거 : .remove(), .pop()
# 업데이트 : .update

# set + for 반복문
for element in set_d:
    print(element)
```

▼ 기타

파이썬의 모든 것은 객체(object)로, 객체는 mutable 하거나 immutable 합니다. 객체가 생성된 후 ID는 변경되지 않으며 해당 객체가 프로그램 실행 중에 유일한 객체라는 것을 보장합니다. (id 함수를 사용하면 객체의 ID를 확인할 수 있습니다.)

- mutable(변경가능) - list, dictionary, set, user-defined class

```
tuple_a = (1, 3, 5, 7, 9)
tuple_a[0] = 2
# type error 발생 !!
# tuple 을 제외한 list, dict, set 타입은 값을 변경할 수 있음

list_a = [2, 4, 6, 8, 10]
print(list_a)
print(id(list_a))
```

```
list_a[1] = 5
print(list_a)
print(id(list_a))
```

- immutable(불변) - int, float, decimal, bool, string, tuple and range

```
num_a = 1
# print(id(num_a))
num_a = 2
print(num_a)
print(id(num_a))
# 값이 바뀐것 처럼 보이지만, id()를 확인하면 메모리상 주소는 같음,
# id() 결과는 컴퓨터 마다 다름.
# 변수에 담긴 객체가 바뀌지 않고 숫자만 바뀐다면, 동일한 id가 나와야 함
```

- sequence - 순서를 가진 container 객체

```
str_a = '가나다라마바사'
tuple_obj = 1, 2, 3, 4, 5
list_obj = [2, 4, 6, 8, 10]

# 인덱싱 - 요소를 가리키는것
print(str_a[0])
print(tuple_obj[1])
print(list_obj[2]

# 슬라이싱 - 요소중 일부를 잘라내는것
print(str_a[0:2])
print(tuple_obj[2:4])
print(list_obj[3:])

# print(str_a[slice(0,2)]) 보다 [start:end:step]등의 슬라이싱이 훨씬 일반적임
```