Python – 자료형

Index

1. 변수

2. 숫자형
   - int
   - float
   - complex

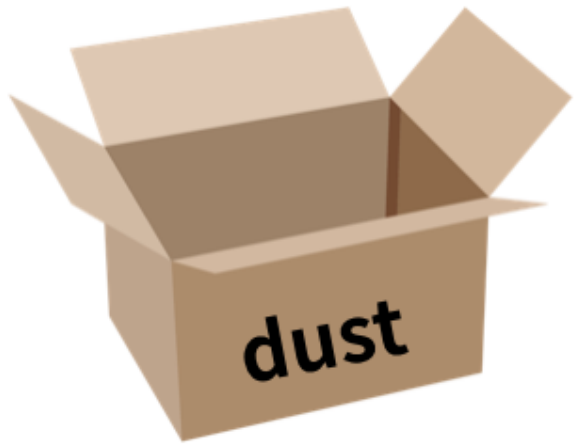
3. Boolean
   - True, False
   - 비교연산자
   - 논리연산자

4. 군집형
   - String
   - List
   - Tuple
   - Dict
   - Set

Variable

변수

박스를 떠올리세요



= 60

dust 는 60이다 (X)

dust 에 60을 저장한다 (O)

Numeric

숫자형

Int (정수형)

- 사칙연산이 가능
- 2진수 0b, 8진수 0o, 16진수 0x로도 표현 가능
- python에서는 long 타입이 없고, 모두 int 타입으로 표기

ex) 1, 3, -2, 5

>> type(10)
결과 : <class 'int'>

float (실수형)

- 사칙연산이 가능 (하지만..?)
- 컴퓨터의 부동소수점 표현 방식 (2진 분수)
- 근삿값, 오차 해결방법? round, 충분히 작은 값 비교, math.isclose(a, b)
- 314e-2

ex) 0.3, 0.5

>> type(6.382)
결과 : <class 'float'>

complex (복소수형)

- 실수부(real)와 허수부(imag)로 구분
- 허수부를 j로 표현

ex) 3 + 5j

>> type(3 - 4j)
결과 : <class 'complex'>

Boolean

불린형

True, False
- 참, 거짓

>> True
결과 : True

>> False
결과 : False

>> type(True)
결과 : <class: bool>

비교연산
- 연산 결과는 **Boolean**

>	>=	<	<=	==	!=
크다	크거나 같다	작다	작거나 같다	같다	같지 않다

>> 3 > 1
결과 : True

>> type(3>1)
결과 : <class: bool>

논리연산
- and : 모두 다 참이면 True 반환
- or : 하나라도 참이면 True 반환
- not : 반대

>> 3 > 1 and 5 < 3
결과 : False

Cluster

군집형

String

- +, * 연산이 가능

```
>> sentence = “Hello, World”

>> multi_line = “””Hello, World
                  Hello, Python”””
+, * 연산
```

- 다양한 문자열 함수

```
>> len(sentence)
결과 : 12

>> sentence.upper()
결과 : 'HELLO, WORLD'

>> sentence.lower()
결과 : 'hello, world'

>> sentence.replace(old, new)
```

String

- 서식지정자
"문자열 %s 문자열"%"추가문자"

>> "I am %s, 29 years old"%"tom"
결과 : 'I am tom, 29 years old'

%(공백)s - 공백 추가

>> "I am %10s, 29 years old"%"tom"
결과 : 'I am tom, 29 years old'

여러 개의 추가 문자

%s - 문자, %d - 정수, %f - 실수
%.2f 소수점 이하 자릿수 설정

>> "I am %s, %s years old"%("tom", "29")
결과 : 'I am tom, 29 years old'

%(공백문자)(개수)s
%03.2f

>> "number = %07.2f"%(3.2323)
결과 : number = 0003.23

- format 함수
''문자열 {0}, {1} 문자열".format(값, 값)

>> "I like {0}, {1} !!".format("apple", banana")
결과 : I like apple, banana !!

{0:(문자)<(숫자)} - 공백추가
- 추가) >, ^

>> "Number {0:a>4} !".format(1)
결과 : Number aaa1 !

List

- 리스트 = [v1, v2, v3]
- 요소 추가 및 변경이 가능

>> fruits = [“apple”, “banana”, “watermelon”]

>> type(fruits)
결과 : <class ‘list’>

Tuple

- 튜플 = (v1, v2, v3)
- 요소 변경이 불가능

>> fruits2 = (“apple”, “banana”, “watermelon”)

>> type(fruits2)
결과 : <class ‘tuple’>

Range 함수

- 연속된 숫자를 생성

>> list(range(3))
결과 : [0, 1, 2]

>> list(range(1, 10, 2))
결과 : [1, 3, 5, 7, 9]

인덱스, 슬라이스

- 인덱스: 배열의 특정 위치
- 슬라이스: 배열의 특정 부분

- (- 인덱스)를 사용가능

```
>> a = [1, 2, 3, 4, 5]
>> a[1]
```
결과 : "2"

```
>> a[0:-1]
```
결과 : [1, 2, 3, 4]

- 증가폭 변경

```
>> a[0:5:2]
```
결과 : [1, 3, 5]

- 끝 인덱스까지 가져오기

```
>> a[2:]
```
결과 : [3, 4, 5]

- 슬라이스 요소 할당

```
>> a[1:4] = [4, 7]
```
결과 : [1, 4, 7, 5]

- 슬라이스 삭제

```
>> a = [1, 2, 3, 4, 5]
>> del a[1:4]
```
결과 : [1, 5]

연산자, 함수 (리스트, 튜플)

- 특정 값 존재여부 확인 in

```
>> a = [1, 2, 3, 4, 5]
>> 1 in a
```
결과 : True

- 반복하기 *

```
>> a*2
```
결과 : [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

- 연결하기 +

```
>> a + [6, 7, 8]
```
결과 : [1, 2, 3, 4, 5, 6, 7, 8]

- 요소개수 구하기 len()

```
>> len(a)
```
결과 : 5

- 리스트 복사 copy()
- 다차원 배열 복사 copy.deepcopy()

```
>> b = a
>> b is a
```
결과 : True
```
>> b = a.copy()
>> b is a
```
결과 : False

리스트 기능

- 요소 추가하기 append(), extend()

```
>> a = [1, 2, 3, 4, 5]
>> a.append(6)
결과 : [1, 2, 3, 4, 5, 6]
>> a.extend([7, 8])
결과 : [1, 2, 3, 4, 5, 6, 7, 8]
```

- 특정 인덱스에 요소 추가 insert(인덱스, 값)

```
>> a = [1, 2, 3, 4, 5]
>> a.insert(2, 100)
결과 : [1, 2, 100, 3, 4, 5]
```

- 특정값의 인덱스 구하기, index(값)

```
>> a = [1, 2, 3, 4, 5]
>> a.index(3)
결과 : 2
```

- 특정값의 개수구하기 count(값)

```
>> a = [1, 1, 1, 2, 2, 3, 3]
>> a.count(1)
결과 : 3
```

- 리스트 요소 삭제 pop(인덱스)

```
>> a = [1, 2, 3, 4, 5]
>> a.pop(2)
결과 : 3
>> a
결과 : [1, 2, 4, 5]
```

- 리스트 특정 값 삭제 remove(값)

```
>> a = [1, 2, 3, 4, 5]
>> a.remove(2)
결과 : [1, 3, 4, 5]
```

- 정렬하기 sort(), sort(reverse=False)

```
>> a = [3, 2, 1, 4]
>> a.sort()
결과 : [1, 2, 3, 4]
>> a.sort(reverse=True)
결과 : [4, 3, 2, 1]
```

- 순서 뒤집기 reverse()

```
>> a = [3, 2, 5, 4, 1]
>> a.reverse()
결과 : [1, 4, 5, 2, 3]
```

Dictionary

- 딕셔너리 = { key1: value1, key2: value2 }
- 요소 추가 및 변경이 가능

>> score = { "name": "sosin", "python": 80, "java" : 30 }
>> score["python"]
결과 : 80

>> score = dict(name="sosin", python= 80, java= 30)

- 키 존재여부 확인 in

>> score = { "name": "sosin", "python": 80, "java" : 30 }
>> "name" in score
결과 : True
>> "age" in score
결과 : False

- 키의 개수 len()

>> len(score)
결과 : 4

- 모든 키, 값 가져오기

>> score.keys()
결과 : dict_keys(["name", "python", "java"])
>> score.values()
결과 : dict_values(["sosin", 80, 30])
>> score.items()
결과 : dict_items([("name", "sosin"), ("python", 80), ("java", 30)])

- 키, 값 추가하기 setdefault(), 수정하기 update()

>> score.setdefault("age", 34)
결과 : { "name": "sosin", "python": 80, "java" : 30, "age": 34 }
>> score.update({"name": "jason"}
결과 : { "name": "jason", "python": 80, "java" : 30, "age": 34 }

- 키 삭제 pop(키, 기본값), 모든 값 삭제 clear()

>> score = { "name": "sosin", "python": 80, "java" : 30 }
>> score.pop("age", 0)
결과 : 0
>> score.clear()
>> score
결과 : {}

- copy(), copy.deepcopy()

Set

- 세트 = { value1, value2, value3 }
- 유니크한, 아이템들의 모임

- 특정 값 존재여부 확인 in

```
>> animals = { "dog", "cat", "monkey" }
>> "cat" in animal
```
결과 : True

- 합집합 |, set.union()

```
>> a = {1,2,3}
>> b = {3,4,5}
>> a | b #set.union(a, b)
```
결과 : {1,2,3,4,5}

- 차집합 -, set.difference()

```
>> a = {1,2,3}
>> b = {3,4,5}
>> a - b #set.difference(a, b)
```
결과 : {1, 2}

- 부분집합 <=, a.issubset(b)

```
>> a = {1,2,3}
>> a <= {1, 2, 3, 4, 5} #a.issubset({1,2,3,4,5})
```
결과 : True

- set()

```
>> set("animal")
```
결과 : { "a", "n", "l", "m", "a", "l" }

- 교집합 &, set.intersection()

```
>> a = {1,2,3}
>> b = {3,4,5}
>> a & b #set.intersection(a, b)
```
결과 : {3}

- 대칭차집합 ^, set.symmetric_difference()

```
>> a = {1,2,3}
>> b = {3,4,5}
>> a ^ b #set. symmetric_difference(a, b)
```
결과 : {1, 2, 4, 5}

- 상위집합 >=, a.issuperset(b)

```
>> a = {1,2,4}
>> a >= {1,2,3} # a.issuperset({1,2,3})
```
결과 : False

Set 기능

– 겹치는 요소 확인 a.isdisjoint(b)

```
>> a = {1,2,4}
>> a.isdisjoint({3, 4, 5})
```

결과 : False

– 추가하기 add()

```
>> a = {1,2,3}
>> a.add(4)
>> a
```

결과 : {1,2,3,4}

– 삭제하기 remove(), discard()

```
>> a = {1,2,3,4}
>> a.remove(1)
>> a
```

결과 : {2,3,4}

```
>> a.discard(3)
```

결과 : {2,4}