컨테이너(Container)

여러 개의 값을 저장할 수 있는 것(~~객체~~)

- 시퀀스(Sequence)형: 순서가 있는(ordered) 데이터
- 비 시퀀스(Non-sequence)형: 순서가 없는(unordered) 데이터

간단 총정리

시퀀스형

list[ ] ----------m=[2,3,4], m[0] = 2, 수정가능하다

tuple( )----------t = (4, 5), t[0] = 4, 수정불가능하다

range( )----------range(n,m,s) : n부터 m-1까지 s step만큼

string ' '--------'a홍길동', s[0] = a

비시퀀스형

set { }---------- 순서보장 X

dictionary {key : value}----------d = {'a' : 'apple'}, key를 통해서 value에 접근한다.

시퀀스(sequence)형 컨테이너

시퀀스 는 데이터가 순서대로 나열된(ordered) 형식을 나타냅니다.

- **주의! 순서대로 나열된 것이 정렬되었다(sorted) 라는 뜻은 아니다.**

특징

1. 순서를 가질 수 있다.
2. **특정 위치의 데이터를 가리킬 수 있다.**

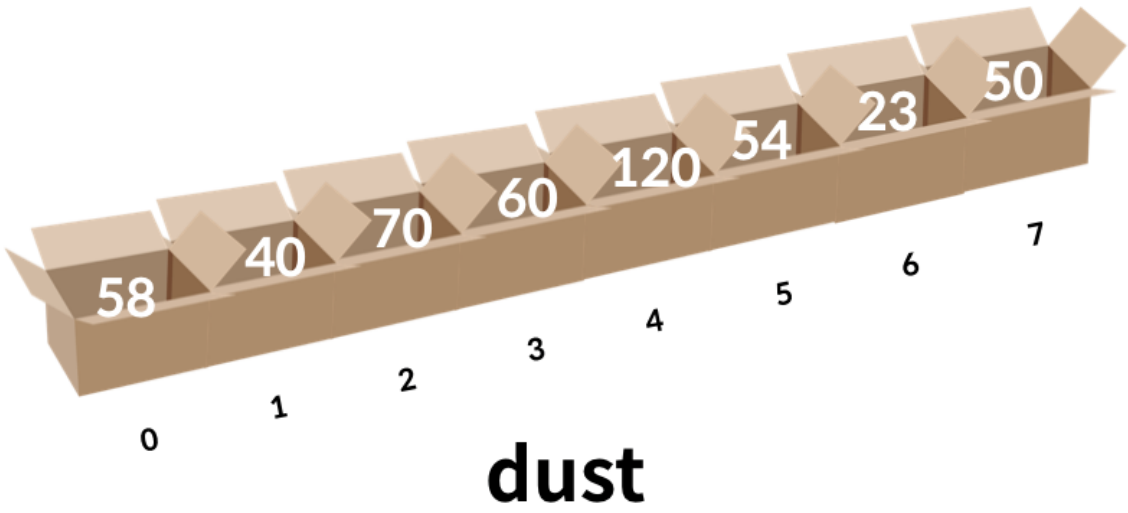
종류

파이썬에서 기본적인 시퀀스 타입은 다음과 같습니다.

- 리스트(list)
- 튜플(tuple)
- 레인지(range)
- 문자형*(string)*

- 바이너리*(binary)* : 따로 다루지는 않습니다.

리스트(list)



활용법

`[value1, value2, value3]`

리스트는 대괄호 `[]` 및 `list()` 를 통해 만들 수 있습니다.

값에 대한 접근은 `list[i]` 를 통해 합니다.

In [ ]:
```
# 빈 리스트를 만들어봅시다.
```

In [ ]:
```
my_list = []
another_list = list()
print(type(my_list))
print(type(another_list))
```

In [ ]:
```
# 원소를 포함한 리스트를 만들어봅시다.
```

In [ ]:
```
lunch = ["카레", "떡갈비"]
print(lunch)
```

In [ ]:
```
# 첫번째 값에 접근해봅시다.
```

In [ ]:
```
lunch[0]
```

In [ ]:
```
word = 'word'
word[:]
# 이거는 문자 전체 나오는 슬라이싱기호이다
```

튜플(tuple)

활용법

`(value1, value2)`

튜플은 리스트와 유사하지만, () 로 묶어서 표현합니다.

그리고 tuple은 수정 불가능(불변, immutable)하고, 읽을 수 밖에 없습니다.

직접 사용하기 보다는 파이썬 내부에서 다양한 용도로 활용되고 있습니다.

In [ ]:
```
# 튜플은 직접 자료구조 활용 X, python 간접적 활용이 높다.
# 왜냐면 수정이 불가능한 속성 때문에
```

In [ ]:
```
# tuple을 만들어봅시다.
```

In [ ]:
```
my_tuple = (1, 2)
print(type(my_tuple))
```

In [ ]:
```
# 아래와 같이 만들 수 있습니다.
```

In [ ]:
```
another_tuple = 1, 2
print(another_tuple)
print(type(another_tuple))
```

In [ ]:
```
# 파이썬 내부에서는 다음과 같이 활용됩니다. (변수 및 자료형 예제에서 사용된 코드입니다.)
```

In [ ]:
```
x, y = 1, 2
print(x, y)
```

In [ ]:
```
# 실제로는 tuple로 처리됩니다.
```

In [ ]:
```

```

In [ ]:
```
# 변수의 값을 swap하는 코드 역시 tuple을 활용하고 있습니다.
```

In [ ]:
```
x = 1
y = 100
x,y = y, x
print(x,y)    # 튜플이 활용되고 있다. 보이지 않는 소괄호들이 있다.
```

In [ ]:
```
# 빈 튜플은 빈 괄호 쌍으로 만들어집니다.
```

In [ ]:
```
empty = ()
```

```
print(type(empty))
print(len(empty))
```

In [ ]:
```
tuple1 = ('hello')
type(tuple1)
```

In [ ]:
```
# 하나의 항목으로 구성된 튜플은 값 뒤에 쉼표를 붙여서 만듭니다.
```

In [ ]:
```
single_tuple = ('hello',)
print(type(single_tuple))
print(len(single_tuple))
```

In [ ]:
```
# 리스트는 특정 원소를 변경할 수 있습니다.
my_list = [1, 3]
my_list[0] = '첫 번째'
print(my_list)
```

In [ ]:
```
# 튜플은 변경이 불가합니다.
my_tuple = (1, 3)
my_tuple[0] = '첫 번째'
print(my_tuple)
```

In [ ]:
```
# 튜플은 읽을 수는 있습니다. 리스트는 더 당연한 이야기다.
print(my_list[0])
print(my_tuple[0])     ##엥 결과값이 왜이래
```

레인지(range())

`range` 는 숫자의 시퀀스를 나타내기 위해 사용됩니다.

기본형 : `range(n)`

> 0부터 n-1까지 값을 가짐

범위 지정 : `range(n, m)`

> n부터 m-1까지 값을 가짐

범위 및 스텝 지정 : `range(n, m, s)`

> n부터 m-1까지 +s만큼 증가한다

In [ ]:
```
# range를 만들어봅시다.
```

In [ ]:
```
range(3)  # 0이상 3미만
```

In [ ]:
```
# range에 담긴 값을 list로 바꿔서 확인해봅시다.
```

In [ ]:
```
list(range(3))
```

In [ ]:
```
# 4 ~ 8까지의 숫자를 담은 range를 만들어봅시다.
```

In [ ]:
```
print(list(range(1,46)))
```

In [ ]:
```
list(range(4,9))
```

In [ ]:
```
# 0부터 -9까지 담긴 range를 만들어봅시다.
```

In [ ]:
```
list(range(0,-10))
```

In [ ]:
```
list(range(0,-10,-1))
```

시퀀스에서 활용할 수 있는 연산자/함수

operation	설명
x in s	containment test
x not in s	containment test
s1 + s2	concatenation
s * n	n번만큼 반복하여 더하기
s[i]	indexing
s[i:j]	slicing
s[i:j:k]	k간격으로 slicing
len(s)	길이
min(s)	최솟값
max(s)	최댓값
s.count(x)	x의 개수

In [ ]:
```
# containment test를 확인해봅시다.
```

In [ ]:
```
s = 'string'
print('a' in s)
```

In [ ]:
```
l = [1, 2, 3, 5, 1]
print(3 in l)
```

In [ ]:
```
# concatenation(연결, 연쇄)를 해봅시다.
```

In [ ]:
```
```

```
print('안녕,' + '하세요')
print((1, 2) + (5, 6))
print([1,2] + [5,6])
```

In [ ]:
```
# 숫자 0이 6개 있는 list를 만들어봅시다.
```

In [ ]:
```
my_list = [0,0,0,0,0,0]
my_list = [0]*6
```

In [ ]:
```
# indexing과 slicing을 하기 위해 list하나를 만들어주세요.
```

In [ ]:
```
location = ['서울', '대전', '구미', '광주']
```

In [ ]:
```
# 두번째, 세번째 값만 가져와봅시다.
```

In [ ]:
```
# indexing
location[0]
```

In [ ]:
```
# slicing
location[1:3]
```

In [ ]:
```
# 0부터 30까지의 숫자를 3씩 증가시킨 리스트로 만들어봅시다.
```

In [ ]:
```
num_list = list(range(0,31))
print(num_list)
```

In [ ]:
```
sample_list = list(range(0,31,3))
print(sample_list)
```

In [ ]:
```
num_list[0:len(num_list):3]
```

In [ ]:
```
num_list[0::3]
```

In [ ]:
```
list(range(0,31,3))
```

In [ ]:
```
# 위에서 만든 list의 길이를 확인해봅시다.
```

In [ ]:
```
len([1,4])
```

In [ ]:
```
# 위에서 만든 list의 최솟값, 최댓값을 확인해봅시다.
```

In [ ]:
```
print(max(num_list))
```

```
print(min(num_list))
```

In [ ]:
```
# list에 담긴 특정한 것의 개수를 확인할 수도 있습니다.
```

In [ ]:
```
a = [1,1,2]
a.count(1)
```

비 시퀀스형(Non-sequence) 컨테이너

- 셋(set)
- 딕셔너리(dictionary)

set

set 은 순서가 없는 자료구조입니다.

- set 은 수학에서의 집합과 동일하게 처리된다.
- set 은 중괄호 {} 를 통해 만들며, 순서가 없고 중복된 값이 없다.
- 빈 집합을 만들려면 set() 을 사용해야 합니다. ({} 로 사용 불가능.)

활용법

```
{value1, value2, value3}
```

In [ ]:
```
# set 두개를 만들어서 연산자들을 활용해봅시다.
```

In [ ]:
```
set_a = {1, 2, 3}
print(set_a)
set_b = {3, 6, 9}
print(set_b)      # 순서가 보장되지 않는다. 그래서 출력결과의 순서가 자기맘대로
```

In [ ]:
```
set_a - set_b
```

In [ ]:
```
# 합집합
set_a | set_b
```

In [ ]:
```
# 교집합
set_a & set_b
```

In [ ]:
```
# set은 중복된 값이 있을 수 없습니다.
```

In [ ]:
```
{1,2,3,1,1}
```

- set 을 활용하면 list 의 중복된 값을 손쉽게 제거할 수 있습니다.

In [ ]:
```
# set으로 중복된 값을 제거해봅시다.
```

In [ ]:
```
list_a = [1, 2, 3, 1, 1, 2]
```

In [ ]:
```
# 다시 list로 바꿔서 확인해봅시다.
```

In [ ]:
```
list(set(list_a))     # 중복이 제거된 리스트가 생긴당! 그러나 순서가 보장되지 않는당...
```

dictionary

dictionary 는 key 와 value 가 쌍으로 이뤄져있으며, 궁극의 자료구조이다.

활용법

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

- {} 를 통해 만들며, dict() 로 만들 수도 있다.
- key 는 **변경 불가능(immutable)한 데이터**만 가능하다. (immutable : string, integer, float, boolean, tuple, range)
- value 는 list , dictionary 를 포함한 모든 것이 가능하다.

In [ ]:
```
# 비어있는 dictionary를 두가지 방법으로 만들어봅시다.
```

In [ ]:
```
dict_a = {}
print(dict_a,type)
dict_b = dict()
print(dict_b)
```

In [ ]:
```
# dictionary는 중복된 key는 존재할 수가 없습니다.
```

In [ ]:
```
my_dict = {'김준호':'남','김준호':'여','홍길동':'남'}    # 중복된 키가 존재 불가. 뒤에꺼 Val
my_dict
```

In [ ]:
```
# 지역번호(서울-02 경기-031)가 담긴 전화번호부를 만들어봅시다.
```

In [ ]:
```
phone = {'서울' : '02', '경기':'031'}
phone
```

In [ ]:
```
# 딕셔너리의 .keys() 메소드를 활용하여 key를 확인 해볼 수 있습니다.
```

In [ ]:
```
phone['서울']
```

In [ ]:
```
# 딕셔너리의 .values() 메소드를 활용하여 value를 확인 해볼 수 있습니다.
```

In [ ]:
```
phone.keys()
```

In [ ]:
```
phone_values = phone.values()
print(phone_values, type(phone_values))   # 유사 리스트일 뿐, 딕셔너리이다.
```

In [ ]:
```
# 딕셔너리의 .items() 메소드를 활용하여 key, value를 확인 해볼 수 있습니다.
```

In [ ]:
```
# dict_items 는 유사 리스트.. 리스트로 변경해서 볼 수 있다.
# (key,value)가 튜플로 묶인 원소들로 만들어져있다!

phone.items()
```

In [ ]:
```
list(phone.items())[0]
```

In [ ]:
```
type(list(phone.items())[0])
```

컨테이너형 형변환

파이썬에서 컨테이너는 서로 변환할 수 있습니다.

	string	list	tuple	range	set	dictionary
string		O	O	X	O	X
list	O		O	X	O	X
tuple	O	O		X	O	X
range	O	O	O		O	X
set	O	O	O	X		X
dictionary	O	O (key만)	O (key만)	X	O (key만)	

In [ ]:
```
# 어떠한 자료 구조도 list, dictionary 교환 불가
```

In [ ]:
```
# list를 형 변환 해봅시다
```

In [ ]:
```
int(3.5)
```

In [ ]:
```
int('3')
```

In [ ]:
```
float('3.5')
```

In [ ]:
```
int('3.5')   # 불가
```

In [ ]:
```
l = [1, 2, 3, 4]
# str(l)
# tuple(l)
# set(l)
# range(l)   # 불가
# dict(l)
```

In [ ]:
```
# tuple을 형 변환 해봅시다
```

In [ ]:
```
t = (1, 2, 3, 4)
# str(t)
# list(t)
# set(t)
# # range(t)
# # dict(t)
```

In [ ]:

```
# range를 형 변환 해봅시다
```

In [ ]:
```
r = range(1, 5)
# str(r)
# list(r)
# set(r)
# tuple(r)
# # dict(r)
```

In [ ]:
```
# set을 형 변환 해봅시다
```

In [ ]:
```
s = {1, 2, 3, 4}
# str(s)
# list(s)
# tuple(s)
# # range(s)
# # dict(s)
```

In [ ]:
```
# dictionary를 형 변환 해봅시다
```

In [ ]:
```
d = {'name': 'ssafy', 'year': 2020}
# str(d)
# list(d)   # 리스트는 키만 모아서 해준다
# tuple(d)  # 튜플도 키만 모아서
# set(d)    # set도 키만 모아서
# # range(d)
# 일반적으로 딕셔너리는 키를 통해서 value에 접근한다. 따라서 키를 알려주면 value를 알 수 있는
```

데이터의 분류

> `mutable` vs. `immutable`

데이터는 크게 변경 가능한 것(`mutable`)들과 변경 불가능한 것(`immutable`)으로 나뉘며, python은 각각을 다르게 다룹니다.

변경 불가능한(`immutable`) 데이터

- 리터럴(literal)
  - 숫자(Number)
  - 글자(String)
  - 참/거짓(Bool)
- range()
- tuple()
- frozenset()

In [ ]:
```
range(1,46)
```

In [ ]:
```
t = (1,3)
t[0] = 2
```

In [ ]:
```
name = '홍길동'
name[0] = '김'    # 바꿀 수 없음. 즉 immutable하다.
```

In [ ]:
```
# mutable 한 예시
l = [1,3]
l[0] = 2
l
```

In [ ]:
```
# immutable 데이터의 복사는 어떻게 이루어질까?
num1 = 20
num2 = num1
num2 = 10

print(num1)
print(num2)
```

In [ ]:
```
%%html
<iframe width="800" height="500" frameborder="0" src="http://pythontutor.com/iframe-embed.ht
```

변경 가능한(mutable) 데이터

- list
- dict
- set

In [ ]:
```
# 같은 곳을 바라보고
# mutable 데이터의 복사는 어떻게 이루어질까?
num1 = [1, 2, 3, 4]
num2 = num1
num2[0] = 100

print(num1)
print(num2)
```

In [ ]:
```
# 서로 다른 곳을 바라보고
num1 = [1,2,3,4]
# 새로운 리스트를 만들어서!!!
num2 = list(num1)
num2[0] = 100

print(num1)
print(num2)
```

In [ ]:
```
%%html
<iframe width="800" height="500" frameborder="0" src="http://pythontutor.com/iframe-embed.ht
```

정리

컨테이너(Container)