List, Tuple, Dictionary dan Set

Struktur Data Super di Python

Husni

Husni.Trunojoyo.ac.id

Outline

- Pendahuluan
- List
- Tuple
- Dictionary
- Set
- Perbandingan
- Struktur data Lebih Besar
- Latihan

Tipe data dasar di Python: booleans, integers, floats dan strings
Jika tipe dasar sebagai atoms, maka struktur data yang akan didiskusikan adalah seperti molekul.

Tipe Data Paling Simpel

- *booleans* (punya nilai True atau False)
- *integers* (bilangan bulat seperti 42 atau 100000000)
- *floats* (bilangan dengan titik decimal seperti 3.14159, atau dengan eksponen 1.0e8 yang sama dengan 100000000.0)
- *strings* (serangkaian karakter teks)

Lists dan Tuples

- Bahasa menyajikan deretan item yang diindeks oleh posisi integer: satu, dua dan seteruskan sampai akhir.
- Contoh umum, *Strings:* sequences of characters
- Dua struktur deret lain: *tuples* dan *lists*
- Elemen dari list dan tuple dapat berbeda tipe, dapat obyek Python apapun.
- Memungkinkan kita membuat struktur sedalam dan sekompleks kebutuhan

Mengapa Python Pakai List dan Tuple?

- Tuples bersifat *immutable*; ketika ditetapkan elemen ke tuple, itu dimatikan dan tidak bisa diubah.
- Lists bersifat *mutable*, artinya kita dapat menyisipkan dan menghapus elemen sesuai kebutuhan.

List

- Bagus untuk menyimpan sesuatu secara urut, terutama saat urutan dan content mungkin berubah.
- lists bersifat mutable. Kita dapat mengubah list di-tempat, menambahkan elemen baru, menghapus atau menimpa elemen yang exist.
- Konten yang sama boleh muncul lebih dari sekali di dalam list

Membuat List dengan [] atau Fungsi list()

- List dibuat dari nol atau lebih elemen, dipisahkan koma, dan dilingkupi oleh kurung siku

```
>>> empty_list = [ ]
>>> weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday']
>>> big_birds = ['emu', 'ostrich', 'cassowary']
>>> first_names = ['Graham', 'John', 'Terry', 'Terry', 'Michael’]

>>> another_empty_list = list()
>>> another_empty_list
```

- Satu cara lain membuat list dinamakan *list comprehension*…Lain waktu ya!

Ubah Tipe Data Lain ke Lists dengan list()

```
>>> list('cat’)
>>> a_tuple = ('ready', 'fire', 'aim’)
>>> list(a_tuple)
```

['c', 'a', 't']

['ready', 'fire', 'aim']

- Memisah String dengan fungsi split()

```
>>> birthday = '1/6/1952’
>>> birthday.split('/’)
```

['1', '6', '1952']

- Bagaimana jika kita mempunyai lebih dari satu string pemisah?

Split()

```
>>> splitme = 'a/b//c/d///e’
>>> splitme.split(‘/’)
```

['a', 'b', '', 'c', 'd', '', '', 'e']

Gunakan pemisah string dua karakter //:

```
>>> splitme = 'a/b//c/d///e’
>>> splitme.split('//')
```

['a/b', 'c/d', '/e']

Mengambil Data Menggunakan [*offset*]

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
>>> marxes[1]
>>> marxes[2]
```

'Groucho'

'Chico'

'Harpo'

Seperti di strings, indeks negative menghitung dari ujung:

```
>>> marxes[-1]
>>> marxes[-2]
>>> marxes[-3]
```

'Harpo'

'Chico'

'Groucho'

Lists dari Lists

Lists dapat berisi elemen berbeda tipe, termasuk list lain

```
>>> small_birds = ['hummingbird', 'finch']
>>> extinct_birds = ['dodo', 'passenger pigeon', 'Norwegian Blue']
>>> carol_birds = [3, 'French hens', 2, 'turtledoves']
>>> all_birds = [small_birds, extinct_birds, 'macaw', carol_birds]
>>> all_birds
>>> all_birds[0]
```

['hummingbird', 'finch']

[['hummingbird', 'finch'], ['dodo', 'passenger pigeon', 'Norwegian Blue'], 'macaw', [3, 'French hens', 2, 'turtledoves']]

Item Pertama dari extinct_birds?

```
>>> all_birds[1]
```

['dodo', 'passenger pigeon', 'Norwegian Blue']

Ekstrak dari all_birds dengan menetapkan dua indeks:

```
>>> all_birds[1][0]
```

'dodo'

Mengubah Item Berdasarkan [*offset*]

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[2] = 'Wanda'
>>> marxes
```

Mengambil Item Berdasarkan Range Offset

```
>>> marxes = ['Groucho', 'Chico,' 'Harpo']
>>> marxes[0:2]
```

Seperti di strings, slices dapat lebih selangkah. Contoh: mulai dari awal lompat 2:

```
>>> marxes[::2]
```

Mulai dari ujung akhir mundur 2 langkah:

```
>>> marxes[::-2]
```

Terakhir, cara membalik isi list:

```
>>> marxes[::-1]
```

Penambahan Item list dengan append()

```
>>> marxes.append('Zeppo')
>>> marxes
```

append() juga dapat menambahkan list:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes.append(others)
>>> marxes
```

Kombinasi List Menggunakan extend() atau +=

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes.extend(others)
>>> marxes
```

Alternatifnya dapat menggunakan +=:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes += others
>>> marxes
```

Offset untuk Tambahkan Item dengan insert()

Fungsi append() menambahkan item hanya ke ujung list

Untuk menambahkan item sebelum offset dalam list, gunakan insert()

```
>>> marxes.insert(3, 'Gummo')
>>> marxes
>>> marxes.insert(10, 'Karl')
>>> marxes
```

Offset: Hapus Item dengan del

Undo (pembatalan) penambahan (*insertion*) terakhir

```
>>> del marxes[-1]
>>> marxes
```

Posisi item lain disesuaikan dan Panjang list berkurang satu:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
>>> marxes[2]
>>> del marxes[2]
>>> marxes
>>> marxes[2]
```

Hapus Item Brdasarkan Value dengan remove()

Jika tidak yakin dimana item di dalam list:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
>>> marxes.remove('Gummo')
>>> marxes
```

Ambil Item berdasarkan Offset dan Hapus Menggunakan pop()

Fungsi pop(): mengambil dan menghapus item daril ist.
Jika disertai offset, item pada offset dikembalikan, jika tidak dianggap -1.
Offset 0: kepala list, pop() atau pop(-1) mengambalikan ekor.

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> marxes.pop()
'Zeppo'
>>> marxes
['Groucho', 'Chico', 'Harpo']
>>> marxes.pop(1)
'Chico'
>>> marxes
['Groucho', 'Harpo']
```

Cari Offset Item Berdasarkan Value dengan index()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> marxes.index('Chico’)
1
```

Test suatu Value di dalam (periksa keberadaan value di dalam list):

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> 'Groucho' in marxes
True
>>> 'Bob' in marxes
False
>>> words = ['a', 'deer', 'a' 'female', 'deer']
>>> 'deer' in words
True
```

Hitung Kemunculan Nilai dengan count()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.count('Harpo')

>>> marxes.count('Bob')

>>> snl_skit = ['cheeseburger', 'cheeseburger’, 'cheeseburger']
>>> snl_skit.count('cheeseburger')
```

Konversi ke String dengan join()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> ', '.join(marxes)
>>> friends = ['Harry', 'Hermione', 'Ron']
>>> separator = ' * '
>>> joined = separator.join(friends)
>>> joined
>>> separated = joined.split(separator)
>>> separated
>>> separated == friends
```

Pengurutan Ulang Item dengan sort()

Mengurutkan item dalam list berdasarkan nilainya: (1) Fungsi list sort() mengurutkan list dengan sendirinya. (2) Fungsi umum sorted() mengembalikan Salinan list terurut.

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> sorted_marxes = sorted(marxes)
>>> sorted_marxes
```

sorted_marxes adalah salinan, dan tidak mengubah list aslinya:

```
>>> marxes
```

Tetapi, fungsi list sort() pada list marxes mengubah marxes:

```
>>> marxes.sort()
>>> marxes
```

Jika tipe data dalam list berbeda, maka secara otomatis dikonversi ke tipe lain:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 4.0]
```

Urutan default adalah ascending, argument reverse=True mengubahnya menjadi descending:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort(reverse=True)
>>> numbers
[4.0, 3, 2, 1]
```

Ambil Panjang Menggunakan len()

len() mengembalikan jumlah item di dalam suatu list:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> len(marxes)
3
```

Penugasan dengan =

Saat suatu list dilewatkan ke variable lain, perubahan terhadap list awal juga berdampak pada variable baru itu

```
>>> a = [1, 2, 3]
>>> a
>>> b = a
>>> b
>>> a[0] = 'surprise'
>>> a
```

Bagaimana dengan b sekarang?

Masih [1, 2, 3], atau ['surprise', 2, 3]?

```
>>> b
```

```
>>> b
['surprise', 2, 3]
>>> b[0] = 'I hate surprises'
>>> b
['I hate surprises', 2, 3]
>>> a
['I hate surprises', 2, 3]
```

Salin dengan copy()

Nilai dari suatu list dapat disalin secara *independent* menggunakan:

- Fungsi list copy()
- Fungsi konversi list()
- Slide list [:]

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> c = list(a)
>>> d = a[:]
```

```
>>> a[0] = 'integer lists are boring'
>>> a
['integer lists are boring', 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]
```

Tuple

Serupa list, tuple merupakan rangkaian item sembarang.

Tetapi, tuple bersifat *immutable*, sehingga kita tidak dapat menambah, menghapus atau mengubah item setelah tuple didefinisikan.

Tuple mirip dengan daftar konstanta.

Membuat Tuple kosong menggunakan ()

```
>>> empty_tuple = ()
>>> empty_tuple
()
```

Membuat Tuple Dengan Satu Atau Lebih Elemen, Setiap Elemen Diikuti Koma

```
>>> one_marx = 'Groucho',
>>> one_marx
('Groucho’,)
```

Jika ada lebih dari satu elemen, ikuti semuanya dengan koma kecuali item terakhir:

```
>>> marx_tuple = 'Groucho', 'Chico', 'Harpo'
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

Membongkar Tuple

Menggunakan tanda kurung lebih dianjurkan

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

Tuple memungkinkan kita melewatkan elemennya ke banyak variable sekaligus:

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> a, b, c = marx_tuple
>>> a
'Groucho'
>>> b
'Chico'
>>> c
'Harpo’
```

Ini disebut *tuple unpacking*.

Tuple untuk Bertukar Nilai dalam Satu Pernyataan

```
>>> password = 'swordfish'
>>> icecream = 'tuttifrutti'
>>> password, icecream = icecream, password
>>> password
'tuttifrutti'
>>> icecream
'swordfish'
```

Fungsi Konversi tuple()

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> tuple(marx_list)
('Groucho', 'Chico', 'Harpo')
```

Tuple vs. List

Mengapa tidak hanya menggunakan list, tidak tuple, untuk semua hal?

- Tuples menggunakan sedikit ruang.
- Kita tidak mungkin salah mengakses item tuple berkali-kali.
- Kita dapat menggunakan tuples sebagai key kamus.
- *Tuple* bernama dapat menjadi alternatif dari obyek.
- Argumen fungsi dilewatkan sebagai tuples.

Dictionary = Kamus

- Kamus ini mirip dengan list, tetapi urutan item tidak dipermasalahkan, dan tidak dipilih menggunakan offset.
- Kita menetapkan suatu *key* unik untuk mengasosiasikan setiap nilai
- Kamus bersifat mutable, sehingga kita dapat menambahkan, menghapus dan mengubah elemen key-value-nya.
- Membuat Kamus dengan {}

```
>>> empty_dict = {}
>>> empty_dict
{}
```

Contoh Kamus

```
>>> bierce = {
... "day": "A period of twenty-four hours, mostly misspent",
... "positive": "Mistaken at the top of one's voice",
... "misfortune": "The kind of fortune that never misses",
... }
>>> bierce
```

Konversi Menggunakan dict()

```
>>> lol = [ ['a', 'b'], ['c', 'd'], ['e', 'f'] ]
>>> dict(lol)
{'c': 'd', 'a': 'b', 'e': 'f'}
```

List dari Tuple dua item:

```
>>> lot = [ ('a', 'b'), ('c',
'd'), ('e', 'f') ]
>>> dict(lot)
{'c': 'd', 'a': 'b', 'e': 'f’}
```

Tuple dari list dua item:

```
>>> tol = ( ['a', 'b'], ['c',
'd'], ['e', 'f'] )
>>> dict(tol)
{'c': 'd', 'a': 'b', 'e': 'f'}
```

List dari string dua karakter:

```
>>> los = [ 'ab', 'cd', 'ef' ]
>>> dict(los)
{'c': 'd', 'a': 'b', 'e': 'f’}
```

Tuple dari string dua karakter:

```
>>> tos = ( 'ab', 'cd', 'ef' )
>>> dict(tos)
{'c': 'd', 'a': 'b', 'e': 'f'}
```

Tambah atau Ubah Item dengan [*key*]

```
>>> pythons = {
... 'Chapman': 'Graham',
... 'Cleese': 'John',
... 'Idle': 'Eric',
... 'Jones': 'Terry',
... 'Palin': 'Michael',
... }
>>> pythons
{'Cleese': 'John', 'Jones': 'Terry', 'Palin': 'Michael’,
 'Chapman': 'Graham', 'Idle': 'Eric’}
```

Tambah atau Ubah Item dengan [*key*]

```
>>> pythons['Gilliam'] = 'Gerry'
>>> pythons
{'Cleese': 'John', 'Gilliam': 'Gerry', 'Palin': 'Michael',
'Chapman': 'Graham', 'Idle': 'Eric', 'Jones': 'Terry’}

>>> pythons['Gilliam'] = 'Terry'
>>> pythons
{'Cleese': 'John', 'Gilliam': 'Terry', 'Palin': 'Michael',
'Chapman': 'Graham', 'Idle': 'Eric', 'Jones': 'Terry'}
```

Kunci Kamus Harus Unik

```
>>> some_pythons = {
... 'Graham': 'Chapman',
... 'John': 'Cleese',
... 'Eric': 'Idle',
... 'Terry': 'Gilliam',
... 'Michael': 'Palin',
... 'Terry': 'Jones',
... }
>>> some_pythons
{'Terry': 'Jones', 'Eric': 'Idle', 'Graham': 'Chapman',
'John': 'Cleese', 'Michael': 'Palin'}
```

Kombinasi Kamus dengan update()

```
>>> pythons = {
... 'Chapman': 'Graham',
... 'Cleese': 'John',
... 'Gilliam': 'Terry',
... 'Idle': 'Eric',
... 'Jones': 'Terry',
... 'Palin': 'Michael’,
... }
>>> others = { 'Marx': 'Groucho', 'Howard': 'Moe’ }
>>> pythons.update(others)
```

```
>>> first = {'a': 1, 'b': 2}
>>> second = {'b': 'platypus'}
>>> first.update(second)
>>> first
{'b': 'platypus', 'a': 1}
```

Hapus Item Berdasarkan Key dengan del

```
>>> del pythons['Marx']
>>> del pythons['Howard’]
```

Hapus semua item dengan clear() atau lewatkan kamus kosong {}

```
>>> pythons.clear()
>>> pythons
{}
>>> pythons = {}
>>> pythons
{}
```

Test Keberadan Key Menggunakan in

```
>>> pythons = {'Chapman': 'Graham', 'Cleese':
'John', 'Jones': 'Terry', 'Palin': 'Michael’}
>>> 'Chapman' in pythons
True
>>> 'Palin' in pythons
True
>>> 'Gilliam' in pythons
False
```

Mengambil Item Berdasarkan [*key*]

```
>>> pythons['Cleese’]
```

Jika key tidak ada dalam Kamus? Error! Hindarinya dengan 2 pendekatan.

Cek key tersebut lebih dulu

```
>>> 'Marx' in pythons
```

Menggunakan fungsi khusus kamus get()

```
>>> pythons.get('Cleese')
'John’
>>> pythons.get('Marx', 'Not a Python')
'Not a Python'
```

Kita menyediakan dictionary,
key, dan optional value.
Jika key tersebut ada, nilainya diperoleh. Jika tidak, diperoleh optional value, jika telah ditetapkan.

Ambil Semua Key Menggunakan keys()

```
>>> signals = {'green': 'go', 'yellow': 'go faster', 'red':
'smile for the camera'}
>>> signals.keys()
dict_keys(['green', 'red', 'yellow’])
```

Ambil semua nilai menggunakan values():

```
>>> list( signals.values() )
['go', 'smile for the camera', 'go faster’]
```

Ambil semua pasangan Key-Value menggunakan items()

```
>>> list( signals.items() )
[('green', 'go'), ('red', 'smile for the camera'), ('yellow',
'go faster')]
```

Penugasan dengan =

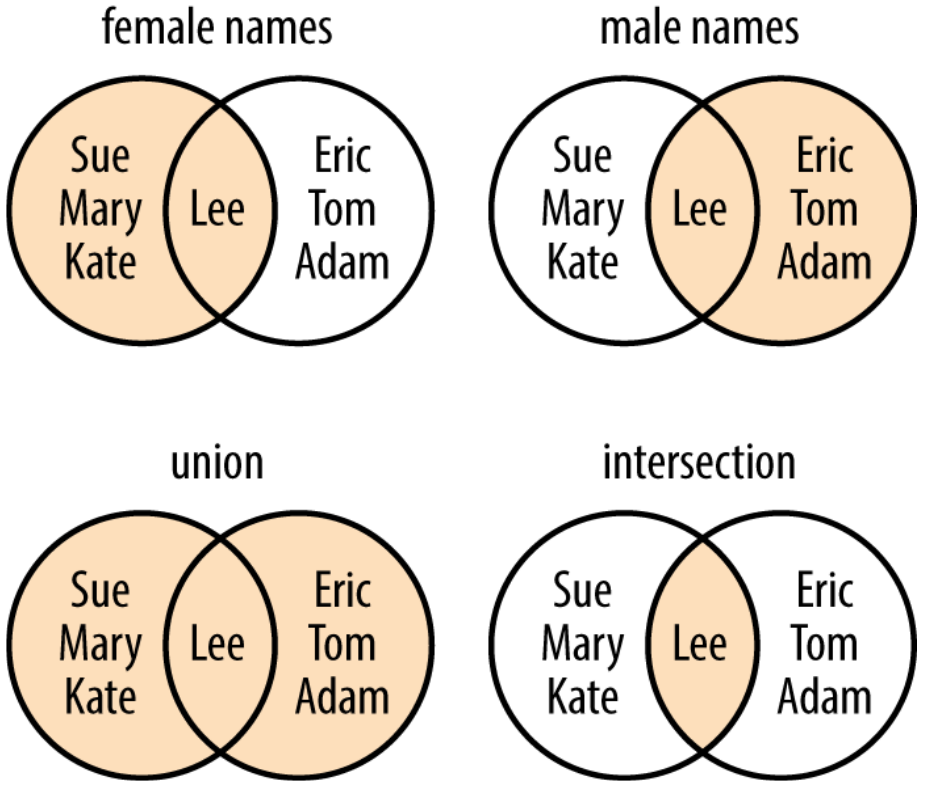
```
>>> signals = {'green': 'go', 'yellow': 'go faster',
'red': 'smile for the camera'}
>>> save_signals = signals
>>> signals['blue'] = 'confuse everyone'
>>> save_signals
{'blue': 'confuse everyone', 'green': 'go',
'red': 'smile for the camera', 'yellow': 'go
faster'}
```

Salin dengan copy()

```
>>> signals = {'green': 'go', 'yellow': 'go faster',
'red': 'smile for the camera'}
>>> original_signals = signals.copy()
>>> signals['blue'] = 'confuse everyone'
>>> signals
{'blue': 'confuse everyone', 'green': 'go',
'red': 'smile for the camera', 'yellow': 'go faster'}
>>> original_signals
{'green': 'go', 'red': 'smile for the camera', 'yellow':
'go faster'}
```

Set = Himpunan

- Set atau himpunan mirip dengan kamus yang tidak mempunyai value, hanya memiliki kunci atau key.



Membuat Set dengan set()

Untuk membuat set, gunakan fungsi set() atau tuliskan satu atau lebih nilai yang dipisahkan koma dalam kurung kurawal.

```
>>> empty_set = set()
>>> empty_set
set()
>>> even_numbers = {0, 2, 4, 6, 8}
>>> even_numbers
{0, 8, 2, 4, 6}
>>> odd_numbers = {1, 3, 5, 7, 9}
>>> odd_numbers
{9, 3, 1, 5, 7}
```

Konversi dari Tipe Data Lain dengan set()

Himpunan dapat dibuat dari suatu list, string, tuple atau dictionary, mengabaikan nilai duplikat.

```
>>> set( 'letters' )
{'l', 'e', 't', 'r', 's’}
```

Membuat Set dari List:

```
>>> set( ['Dasher', 'Dancer', 'Prancer', 'Mason-Dixon'] )
{'Dancer', 'Dasher', 'Prancer', 'Mason-Dixon’}
```

Set dari Tuple:

```
>>> set( ('Ummagumma', 'Echoes', 'Atom Heart Mother') )
{'Ummagumma', 'Atom Heart Mother', 'Echoes’}
```

Jika dari Dictionary, hanya gunakan keys:

```
>>> set( {'apple': 'red', 'orange': 'orange', 'cherry': 'red'} )
{'apple', 'cherry', 'orange'}
```

Test Keberadaan Nilai Menggunakan in

Membuat Dictionary bernama **drabcd**

```
>>> drabcd = {
... ‘marxyz': {‘voabc', ‘veabc'},
... ‘blaxyz': {‘voabc', ‘kaabc'},
... ‘whixyz': {‘crabc', ‘kaabc', ‘voabc'},
... ‘manxyz': {‘ryabc', ‘veabc', ‘biabc'},
... ‘scrxyz': {‘orabc', ‘voabc'}
... }
```

Mana `drabcd` yang mengandung `voabc`?

Mana **drabcd** yang mengandung voabc?

```
>>> for name, contents in drabcd.items():
... if 'voabc' in contents:
... print(name)
...
scrxyz
marxyz
blaxyz
whixyz
```

Bagaimana mendapatkan `voabc` tetapi tidak `veabc` dan juga tidak `crabc`?

```
>>> for name, contents in drabcd.items():
... if ‘voabc' in contents and not (‘veabc' in contents or
... ‘crabc' in contents):
... print(name)
...
scrxyz
blaxyz
```

Kombinasi dan Operator

- Anda ingin memeriksa kombinasi himpunan nilai?
- Bagaimana jika ingin diperoleh drabcd yang mempunyai orabc atau veabc?
- Gunakan operator *set intersection* (&)

```
>>> for name, contents in drabcd.items():
... if contents & {‘veabc', ‘orabc'}:
... print(name)
...
scrxyz
marxyz
manxyz
```

Kombinasi dan Operator

- voabc tetapi bukan crabc atau veabc

```
>>> for name, contents in drabcd.items():
... if ‘voabc' in contents and not contents & {‘veabc', ‘crabc'}:
... print(name)
...
scrxyz
blaxyz
```

Kombinasi dan Operator

```
>>> satu = drabcd[‘blaxyz']
>>> dua = drabcd[‘whixyz']

>>> a = {1, 2}
>>> b = {2, 3}

>>> a & b
{2}

>>> a.intersection(b)
{2}

>>> satu & dua
{‘kaabc', ‘voabc'}
```

```
>>> a | b
{1, 2, 3}
>>> a.union(b)
{1, 2, 3}



>>> satu | dua
{‘crabc’, ‘kaabc', ‘voabc'}
```

Kombinasi dan Operator

```
>>> a - b
{1}
>>> a.difference(b)
{1}
>>> satu - dua
set()
>>> dua - satu
{'cream'}

>>> a ^ b
{1, 3}
>>> a.symmetric_difference(b)
{1, 3}

>>> satu ^ dua
{'cream'}

>>> a <= b
False
>>> a.issubset(b)
False
>>> satu <= dua
True

>>> a <= a
True
>>> a.issubset(a)
True

>>> a < b
False
>>> a < a
False
>>> satu < dua
True

>>> a >= b
False
>>> a.issuperset(b)
False
>>> dua >= satu
True
```

Kombinasi dan Operator

```
>>> a >= a
True
>>> a.issuperset(a)
True

>>> a > b
False
>>> dua > satu
True

>>> a > a
False
```

Perbandingan Struktur Data

- Review: kita membuat list menggunakan kurung siku ([]), tuple dengan koma atau kurung, dan kamus dengan kurung kurawal ({}).
- Di setiap kasus, kita mengakses elemen tunggal dengan kurung siku

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> marx_tuple = 'Groucho', 'Chico', 'Harpo'
>>> marx_dict = {'Groucho': 'banjo', 'Chico': 'piano', 'Harpo': 'harp'}
>>> marx_list[2]
'Harpo'
>>> marx_tuple[2]
'Harpo'
>>> marx_dict['Harpo']
'harp'
```

Untuk list dan tuple, value antara kurung siku adalah offset integer. Untuk kamus, itu key. Untuk ketiganya, hasilnya adalah value.

Struktur Data Lebih Besar

- Kita telah bekerja dengan tipe data booleans, numbers dan strings, juga list, tuple, set dan dictionary.
- Kombinasi dapat digunakan untuk membangun struktur yang lebih besar dan kompleks.

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> pythons = ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']
>>> stooges = ['Moe', 'Curly', 'Larry']
```

Struktur Data Lebih Besar

Kita dapat membuat tuple yang berisi setiap list sebagai elemennya:

```
>>> tuple_of_lists = marxes, pythons, stooges
>>> tuple_of_lists
(['Groucho', 'Chico', 'Harpo'],
['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin'],
['Moe', 'Curly', 'Larry'])
```

Dan kita dapat membuatkan suatu list yang mengandung tiga list:

```
>>> list_of_lists = [marxes, pythons, stooges]
>>> list_of_lists
[['Groucho', 'Chico', 'Harpo'],
['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin'],
['Moe', 'Curly', 'Larry']]
```

Struktur Data Lebih Besar

Terakhir, mari kita buat dictionary dari list. Kita gunakan nama grup komedi sebagai key dan daftar anggotanya sebagai value:

```
>>> dict_of_lists = {'Marxes': marxes, 'Pythons': pythons, 'Stooges': stooges}
>>> dict_of_lists
{'Stooges': ['Moe', 'Curly', 'Larry'],
'Marxes': ['Groucho', 'Chico', 'Harpo'],
'Pythons': ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']}
```

Struktur Data Lebih Besar

- Batasannya ada pada tipe data sendiri. Sebagai contoh, dictionary keys perlu bersifat *immutable*, sehingga suatu list, dictionary atau set tidak dapat menjadi key bagi dictionary lain.
- Tetapi tuple boleh. Sebagai contoh, kita dapat mengindeks *sites of interest* dengan koordinat GPS (latitude, longitude, dan altitude):

```
>>> houses = {
(44.79, -93.14, 285): 'My House',
(38.89, -77.03, 13): 'The White House'
}
```

Latihan

1. Buatlah suatu list bernama years_list, dimulai dengan tahun kelahiran anda, dan seterusnya sampai tahun saat anda berumur 5 tahun. Sebagai contoh, jika anda lahir pada 1980. List years_list = [1980, 1981, 1982, 1983, 1984, 1985].

   - Pada taun berapakah dalam years_list anda berumur 3 tahun?
   - Pada tahun berapakah dalam years_list anda paling tua?

2. Buatlah suatu list bernama things dengan 3 string ini sebagai elemennya: "mozzarella", "cinderella", "salmonella".

   - Huruf besarkan elemen dalam things yang merujuk ke seseorang dan kemudian cetak list tersebut. Apakah itu mengubah elemen di dalam list?
   - Buatlah elemen “cheesy” dari things menjadi huruf besar dan kemudian cetak list.
   - Hapus elemen “disease” dari things dan cetak list.

Latihan

3. Buatlah suatu list bernama surprise dengan elemen: "Groucho", "Chico", and "Harpo".
    - Huruf-kecilkan elemen terakhir dari list surprise, balik hurufnya, dan kemudian besarkan.

4. Buatlah suatu kamus English-to-Indonesia bernama e2i dan cetaklah. Kata-kata awal yang harus ada: dog = anjing, cat = kucing, dan tiger = macan.
    - Menggunakan kamus tiga kata e2i, cetak kata Indonesia dari tiger.
    - Buatlah kamus Indonesia-to-English bernama i2e dari e2i.
    - Gunakan i2e, cetak kata English yang Indonesianya dalah kucing.
    - Buat dan cetak sehimpunan kata English dari kunci dalam e2i.

Latihan

5. Buatlah suatu kamus multilevel bernama life. Gunakan string ini untuk kunci level tertinggi: 'animals', 'plants', dan 'other'. Buatlah kunci 'animals' merujuk ke kamus lain dengan kunci 'cats', 'octopi', dan 'emus'. Buatlah kunci 'cats' merujuk ke suatu list string dengan nilai 'Henri', 'Grumpy', dan 'Lucy'. Buatlah semua kunci lain merujuk ke kamus kosong.

   - Cetak kunci top-level dari life.
   - Cetak kunci untuk life['animals'].
   - Cetak nilai untuk life['animals']['cats'].