Struktur Data & Pemrograman Fungsional

Ali Akbar Septiandri

Universitas Al-Azhar Indonesia

aliakbars@live.com

February 27, 2017

Overview

1. Struktur Data
   - Struktur Data Bawaan
   - Stack & Queue
2. Pemrograman Fungsional
   - Fungsi Lambda
   - Higher-Order Functions
   - Iterator & Generator

Pengumuman

1. Segera *enroll* di e-learning UAI
2. Pengumpulan tugas akan melalui e-learning
3. Materi secara lebih lengkap tetap akan disampaikan melalui `http://uai.aliakbars.com/python.html`
4. Akan ada Google Code Jam 2017 dalam beberapa hari lagi - segera daftar!

Struktur Data

Abstract Data Type

Struktur data di Python memiliki ADT bawaan yang umum ada di bahasa pemrograman lain, di antaranya [Python Software Foundation, 2017]:

1. list
2. set
3. map (dictionary)
4. stack
5. queue

List

Beberapa karakteristik yang dimiliki oleh *list* dalam bahasa Python:

1. merupakan array yang dapat dikembangkan penggunaannya sebagai struktur data yang lain, misalnya *stack* atau *queue*
2. sejatinya bukan merupakan *linked list*
3. bersifat *mutable*
4. *linked list* dapat ditemukan sebagai *generator*

Operasi pada List

- `list.append(x)`
- `list.extend([x])`
- `list.insert(i, x)`
- `list.remove(x)`
- `list.pop([i])`
- `list.index(x)`
- `list.count(x)`
- `list.sort(cmp=None, key=None, reverse=False)`
- `list.reverse()`

Contoh Operasi pada List

Example

```
>>> x = range(10)
>>> x.extend(range(3))
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
>>> x.append(5)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 5]
>>> x.count(0)
2
```

Pertanyaan

Jika kita memiliki *list* `x`:

1. Apa bedanya `x.sort()` dengan `sorted(x)`?

Pertanyaan

Jika kita memiliki *list* `x`:

1. Apa bedanya `x.sort()` dengan `sorted(x)`?
2. Apa bedanya `x[::-1]` dengan `x.reverse()`?

List of Characters

1. Ingat, string di Python dapat dilihat sebagai *list of characters*!

List of Characters

1. Ingat, string di Python dapat dilihat sebagai *list of characters*!
2. Namun, beberapa metodenya dihilangkan, e.g. `sort()` dan `reverse()`

List of Characters

1. Ingat, string di Python dapat dilihat sebagai *list of characters*!
2. Namun, beberapa metodenya dihilangkan, e.g. `sort()` dan `reverse()`
3. Perbandingan *list* di Python menggunakan *lexicographical ordering*

Operasi pada String

- `str.split('')`
- `str.find('')`
- `str.join([])`
- `str.replace(old, new)`
- `str.count('')`
- `str.upper()`
- `str.lower()`

List Comprehensions

List dapat dibentuk dengan lebih ringkas jika didasarkan pada *list* lain sebelumnya. Contoh berikut adalah pembuatan *list* hasil kuadrat dari bilangan-bilangan dalam sebuah *list*.

Example (Konvensional)

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Example (List comprehensions)

```
>>> squares = [x**2 for x in range(10)]
```

Tuple

1. Jauh lebih sederhana dari *list*, tidak banyak metode yang tersedia
2. Bersifat *immutable*
3. Biasa digunakan sebagai bentuk kembalian dari basis data
4. Ditulis menggunakan tanda kurung, e.g. `(1, 3, 5)`

Set

Set atau himpunan memiliki elemen bernilai unik dan tidak mendukung *indexing*. Beberapa metode bawaannya antara lain:

- `set.intersection()` (operator &)
- `set.union()` (operator |)
- `set.difference()` (operator -)
- `set.issubset()` (operator ^)

Contoh Set

Example

```
>>> a = range(5)
>>> b = [1, 3, 5]
```

Evaluasi hasil dari:

1. `a.intersection(b)`
2. `a.union(b)`
3. `a.difference(b)`
4. `b.difference(a)`
5. `b.issubset(a)`

Map/Dictionary

1. ADT yang berbentuk pasangan *key-value*
2. Bersifat *mutable*
3. Dapat diinisialisasi dengan `dict()` atau {}

Map/Dictionary

Beberapa metode bawaan *dictionary*:

- `dict.keys()`
- `dict.values()`
- `dict.items()`
- `dict.pop('')`
- `dict.update({})`

Contoh Dictionary

Example

```
>>> x = {'a': 5, 'b': 3, 'c': 10}
>>> x.keys()
['a', 'b', 'c']
>>> x.values()
[5, 3, 10]
>>> x.items()
[('a', 5), ('c', 10), ('b', 3)]
>>> x.pop()
5
>>> x.update({'a': 9, 'b': 2})
>>> x
{'a': 9, 'c': 10, 'b': 2}
```

Stack

List dalam Python dapat difungsikan sebagai *stack* dengan menggunakan metode `append()` (sebagai pengganti *push*) dan `pop()`.

Example

```
>>> stack = [1, 6, 5]
>>> stack.append(10)
>>> stack
[1, 6, 5, 10]
>>> stack.pop()
10
>>> stack.pop()
5
```

Queue

Untuk memfungsikan *list* sebagai *queue*/antrean, perlu menggunakan modul `collections.deque`

Example ([Python Software Foundation, 2017])

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

Pemrograman Fungsional

Pemrograman Fungsional dengan Python

Sebagai bahasa yang multiparadigma, Python dapat digunakan sebagai bahasa pemrograman dengan paradigma fungsional. Beberapa keuntungan pemrograman fungsional [Kuchling, 2006]:

1. Pembuktian formal (matematis)
2. Modularitas tinggi
3. Mempermudah *debugging* dan pengujian
4. *Composability*

Fungsi Lambda

1. Merupakan fungsi anonim, i.e. tidak memiliki nama atau identitas
2. Dapat dijadikan argumen untuk *higher-order functions*
3. Juga banyak digunakan di JavaScript

Contoh Fungsi Lambda

Example (Konvensional)

```
def f(x):
    return x**2 + 10
```

Example (Fungsi lambda)

```
f = lambda x: x**2 + 10
```

Higher-Order Functions

Fungsi yang mengembalikan fungsi lain atau menerima fungsi lain sebagai argumen disebut sebagai *higher-order functions*.

Example (high.py)

```
def f(x, g):
    return g(x) + 10

def kuadrat(x):
    return x ** 2

def kubik(x):
    return x ** 3

print f(3, kuadrat)
print f(3, kubik)
print f(3, lambda x: x ** 4)
```

Parameter Opsional

Fungsi di Python dapat menerima parameter bawaan sehingga menjadikannya opsional ketika fungsinya dipanggil.

Example

```
>>> def f(x, e=0.01):
...     return x ** 3 + e
...
>>> f(3)
27.01
>>> f(3, 8)
35
```

filter(), map(), max(), min(), sorted()

Beberapa fungsi bawaan Python merupakan *higher-order functions* dan sebagian menggunakan parameter opsional di dalamnya.

Example (filter)

```
>>> def f(x): return x % 3 == 0 or x % 5 == 0
...
>>> filter(f, range(2, 25))
[3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24]
```

Example (map)

```
>>> def cube(x): return x**3
...
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

filter(), map(), max(), min(), sorted()

Example (min & max)

```
>>> x = [(0, 3, 7), (9, 1, 3), (1, 8, 2)]
>>> max(x)
(9, 1, 3)
>>> max(x, key=lambda z: z[1])
(1, 8, 2)
>>> min(x)
(0, 3, 7)
>>> min(x, key=lambda z: z[2])
(1, 8, 2)
```

filter(), map(), max(), min(), sorted()

Example (min & max)

```
>>> x = [(0, 3, 7), (9, 1, 3), (1, 8, 2)]
>>> max(x)
(9, 1, 3)
>>> max(x, key=lambda z: z[1])
(1, 8, 2)
>>> min(x)
(0, 3, 7)
>>> min(x, key=lambda z: z[2])
(1, 8, 2)
```

Example (sorted)

```
>>> sorted(x, key=lambda z: z[2])
?
```

Iterator

Definisi

"An iterator is an object representing a stream of data; this object returns the data one element at a time." [Kuchling, 2006]

Iterator

Definisi

"An iterator is an object representing a stream of data; this object returns the data one element at a time." [Kuchling, 2006]

Singly linked list

Iterator dapat dilihat sebagai *singly linked-list* karena harus mempunyai metode `next()`

Iterator

Definisi

"An iterator is an object representing a stream of data; this object returns the data one element at a time." [Kuchling, 2006]

Singly linked list

Iterator dapat dilihat sebagai *singly linked-list* karena harus mempunyai metode `next()`

Looping dengan iterator

Setiap pemanggilan `for i in obj` sebetulnya membuat `obj` tersebut menjadi iterator

Contoh Iterator

Example

```
>>> L = [1,2,3]
>>> it = iter(L)
>>> print it
<...iterator object at ...>
>>> it.next()
1
>>> it.next()
2
>>> it.next()
3
>>> it.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
StopIteration
>>>
```

Generator

Definisi

"Generators are a special class of functions that simplify the task of writing iterators. Regular functions compute a value and return it, but **generators return an iterator that returns a stream of values**." [Kuchling, 2006]

Generator

Definisi

"Generators are a special class of functions that simplify the task of writing iterators. Regular functions compute a value and return it, but **generators return an iterator that returns a stream of values**." [Kuchling, 2006]

Example

```
def generate_ints(N):
    for i in range(N):
        yield i
```

Contoh Generator

Example

```
>>> gen = generate_ints(3)
>>> gen
<generator object generate_ints at ...>
>>> gen.next()
0
>>> gen.next()
1
>>> gen.next()
2
>>> gen.next()
Traceback (most recent call last):
  File "stdin", line 1, in ?
  File "stdin", line 2, in generate_ints
StopIteration
```

Kuis

Kuis

Minimum Scalar Product (Google Code Jam Round 1A 2008)
You are given two vectors $v_1 = (x_1, x_2, ..., x_n)$ and $v_2 = (y_1, y_2, ..., y_n)$. The scalar product of these vectors is a single number, calculated as $x_1y_1 + x_2y_2 + ... + x_ny_n$.
Suppose you are allowed to permute the coordinates of each vector as you wish. Choose two permutations such that the scalar product of your two new vectors is the smallest possible, and output that minimum scalar product.

Kuis

Example (Input)

```
2
3
1 3 -5
-2 4 1
5
1 2 3 4 5
1 0 1 0 1
```

Example (Output)

```
Case #1: -25
Case #2: 6
```

References

Python Software Foundation (2017)

The Python Tutorial - 5. Data Structures

https://docs.python.org/2/tutorial/datastructures.html

A. M. Kuchling (2006)

Functional Programming HOWTO

https://docs.python.org/2/howto/functional.html

Terima kasih