Modul 3

Collections, Arrays, and Linked Structures

Ketika melakukan pemrograman komputer, kita akan memerlukan 'tempat nilai' atau 'tempat data'. Umumnya kita sebut 'variabel'. Misal `x = 'Joko'`, `y = 15`.

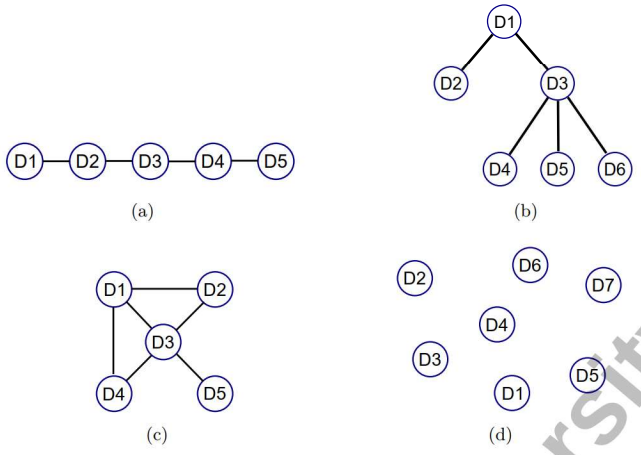
Variabel-variabel[1] ini, sebagai wadah data, mempunyai 'jenis'. Tergantung 'jenis barang' yang disimpannya. Di pelajaran-pelajaran terdahulu sudah kita lihat beberapa tipe data di Python, seperti

- `int` untuk menyimpan bilangan bulat
- `float` untuk menyimpan bilangan pecahan
- `str` untuk menyimpan untaian huruf
- `bool` untuk menyimpan "benar atau salah". `True` atau `False`.
- `list` untuk menyimpan kelompok objek-objek secara urut. List bersifat mutable.
- `tuple` sama seperti list tapi immutable
- `dict` sama seperti list tapi kunci-(index-)nya tidak harus angka
- `set` untuk menyimpan himpunan objek-objek yang unik. Tidak ada objek yang sama di suatu set.

Tipe-tipe data di atas adalah tipe-tipe data dasar. Kita bisa membuat tipe data yang lain sesuai keperluan. Yang perlu diperhatikan ketika membuat tipe data baru:

- Jenis data dasar apa saja yang perlu disimpan di tipe data baru itu?
- Bagaimanakah perilaku tipe data itu?
- Seperti apakah *interface* antara tipe data itu dengan kode yang memanggilnya?

[1] Bahasa Inggris: *variable*, dari *vary-able*. Bermakna kurang lebih 'mampu-diubah'.



Gambar 3.1: Jenis-jenis koleksi.

Di pelajaran tentang `class` pekan lalu kita sudah menyinggung pembuatan, *well*, `class`. Suatu class bisa menjadi prototipe suatu tipe data baru. Ketika di-instantiasi, class itu akan diwujudkan menjadi objek-objek, seperti halnya objek-objek lain[2] di Python.

Tipe data baru yang dibuat oleh programmer lewat sebuah class disebut juga *Abstract Data Type* (ADT).

3.1 Pengertian *Collections*

Collections, diindonesiakan menjadi 'koleksi', adalah kumpulan objek-objek yang kemudian diacu sebagai entitas tunggal. ***Kita sudah pernah menemui contoh-contoh untuk ini sebelumnya*** dalam bentuk `list`, `tuple`, dan `dict`. Tipe `str` bisa juga dianggap sebagai koleksi (suatu string adalah kumpulan huruf-huruf).

Jenis-jenis koleksi

Koleksi bisa dibagi menjadi beberapa jenis jika dilihat dari strukturnya dan bagaimana tiap elemen berkait dengan elemen yang lain. Lihat Gambar 3.1.

- **Koleksi linear**. Item-item yang ada di sebuah *linear collection* diurutkan oleh posisi, seperti ditunjukkan di Gambar 3.1(a). Contoh: daftar belanja, tumpukan piring, dan antrian pelanggan di Bank.
- **Koleksi hirarkis**. Item-item yang terdapat di sebuah *hierarchical collection* diurutkan pada sebuah struktur yang mirip pohon terbalik. Setiap elemen data kecuali yang paling atas memiliki satu orangtua dan memiliki potensi banyak anak. Contoh: sistem direktori

[2]Di Python, semuanya adalah adalah objek (bahkan termasuk fungsi)

berkas di komputer, struktur organisasi di suatu perusahaan, daftar isi buku, struktur berkas XML, struktur data di LDAP, struktur domain internet. Lihat Gambar 3.1(b)

- **Koleksi graf**. Sebuah *graph collection*, disebut juga sebuah *graph*, adalah sebuah koleksi di mana antar item dihubungkan dengan 'pertetanggaan'. Contoh: rute jalanan antar kota, diagram pengkabelan listrik di suatu gedung, hubungan pertemanan di Facebook. Gambar 3.1(c) mengilustrasikannya.
- **Koleks tak urut**. Koleksi tak urut – *unordered collection* – adalah, seperti namanya, koleksi yang tidak memiliki urutan tertentu. Misalnya sekantung kelereng seperti diilustrasikan di Gambar 3.1(d).

Operasi pada koleksi

Seperti sudah diungkapkan di atas, kita sebaiknya mendefinisikan operasi-operasi apa saja yang bisa dilakukan pada suatu koleksi.

- Pencarian dan pengambilan nilai
- Penghapusan
- Penyisipan
- Penggantian
- Pengunjungan tiap elemen (traversal)
- Uji kesamaan
- Menentukan ukuran
- Menyalin

Kamu sudah mengenal tipe data `set`, `list`, `tuple`, `str`, dan `dict`. Termasuk jenis koleksi apakah menurutmu masing-masing tipe data itu? Apakah tiap-tiap tipe data itu mempunyai semua operasi yang diperlukan di atas?

3.2 Array dan Array Dua Dimensi

Array, salah satu bentuk koleksi linear, adalah sebuah struktur data yang diakses atau diganti berdasar pada posisi index. Pada Python, umumnya pemrogram akan memakai tipe data `list` saat memerlukan struktur array. Periksalah bahwa semua operasi koleksi yang diperlukan di atas sudah disediakan oleh Python untuk tipe data `list`.

Array dua dimensi pada prinsipnya bisa dibangun dengan array satu dimensi di mana setiap elemennya adalah array satu dimensi.

Latihan 3.1 Matrix sebagai array dua dimensi. Di bawah ini kita membuat 'matrix' 2×2 dan mengakses beberapa elemennya

```
>>> A = [ [2,3], [5,7] ]
>>> A[0][1]
3
>>> A[1][1]
7
```

Perhatikan bahwa kita mengakses elemen baris `i` kolom `j` dengan cara `m[i][j]`. (Dengan asumsi kita menghitung dari 0). □

Latihan 3.2 Membuat matrix 3×3 berisi 0 semua.

```
>>> B = [ [0 for j in range(3)] for i in range(3) ]
>>> B
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Hey, bagaimana bisa seperti itu? Sudah saatnya kita berkenalan dengan *list comprehension*. □

List Comprehension

List comprehension adalah sebuah cara singkat untuk membuat sebuah list dari sebuah list (atau sesuatu yang iterable lain semisal tuple atau string.). Bentuk dasarnya adalah:
`[`*`expression`* `for` *`iter_var`* `in` *`iterable`*`]`
atau versi yang lebih lanjutnya:
`[`*`expression`* `for` *`iter_var`* `in` *`iterable`* `if` *`condition`*`]`
Beberapa contoh akan membuatnya jelas (cobalah!).

- Membuat list kuadrat bilangan dari 0 sampai 6

  ```
  >>> [x**2 for x in range(0,7)]
  [0, 1, 4, 9, 16, 25, 36]
  ```

- Membuat list yang berisi tuple pasangan bilangan dan kuadratnya, dari 0 sampai 6

  ```
  >>> [(x,x**2) for x in range(7)]
  [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36)]
  ```

- Membuat list kuadrat bilangan genap antara 0 dan 15

  ```
  >>> [x**2 for x in range(15) if x%2==0]
  [0, 4, 16, 36, 64, 100, 144, 196]
  ```

- Membuat list sepanjang 5 elemen yang berisi bilangan 3

  ```
  >>> [3 for i in range(5)]
  [3, 3, 3, 3, 3]
  ```

- Membuat list sepanjang tiga elemen yang berisi list sepanjang 3 elemen angka 0

  ```
  >>> [ [0 for j in range(3)] for i in range(3) ]
  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
  ```

  Itu adalah yang kita buat di contoh pertama kita tentang matrix.

- Membuat matrix identitas 3×3

```
>>> [ [ 1 if j==i else 0 for j in range(3) ] for i in range(3) ]
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

- Membuat list yang berisi huruf vokal suatu string

```
>>> d = "Yogyakarta dan Surakarta."
>>> [x for x in d if x in "aiueoAIUEO"]
['o', 'a', 'a', 'a', 'a', 'u', 'a', 'a', 'a']
```

- Membuat list bilangan prima[a] dari 20 sampai 50

```
>>> [x for x in range(20,50) if apakahPrima(x)]
[23, 29, 31, 37, 41, 43, 47]
```

[a]Kamu harus memuat fungsi `apakahPrima()` yang sudah kamu buat di Modul 1 ke ruangnama lokal.

Dengan pengetahuan di atas, sekarang kamu bisa membayangkan (dan membuat!) fungsi-fungsi

- untuk memastikan bahwa isi dan ukuran matrix-nya konsisten (karena tiap anggota dari list-luar-nya bisa saja mempunyai ukuran yang berbeda-beda, dan bahkan bisa saja berbeda tipe!)[3],
- untuk mengambil ukuran matrixnya,
- untuk menjumlahkan dua matrix,
- untuk mengalikan dua matrix,
- untuk menghitung determinan sebuah matrix bujursangkar.

Bahkan, kamu bisa membuat sebuah `class` yang mewakili matrix.

3.3 Linked Structures

Linked structures, bolehlah kita terjemahkan menjadi *struktur bertaut*[4], adalah salah sebuah struktur data di mana antar elemennya dihubungkan lewat suatu referensi. Struktur berkait ini berisi koleksi objek yang disebut ***simpul***[5] yang tiap-tiap simpulnya mempunyai data dan setidaknya satu tautan ke simpul yang lain.

Struktur bertaut ini bisa menjadi alat untuk membuat koleksi linear, hirarkis, maupun graf. Bentuk linked structure yang paling sederhana adalah linked list.

Linked List

Linked list dibuat dengan menggandengkan satu simpul dengan simpul lain. Lihat kembali Gambar 3.1(a). Sebelum membuat linked list, kita buat dulu wadah-nya menggunakan `class`. Perlu diperhatikan bahwa dalam linked list, setidaknya ada dua hal yang harus ada di setiap objek simpul:

[3]Ini adalah salah satu kelebihan Python yang harus diwaspadai. Di kebanyakan bahasa-bahasa lain, hal ini tidak memungkinkan

[4]Mungkin ada yang lebih memilih *struktur berkait*, *struktur berikat*, *struktur berantai*, atau *struktur bergandeng*.

[5]Bahasa Inggris: ***node***

- Muatannya, atau istilahnya *cargo*-nya. Kadang disebut 'data'.
- Penambat- atau kait-nya. Ini yang akan dipakai untuk menunjuk ke objek yang lain.

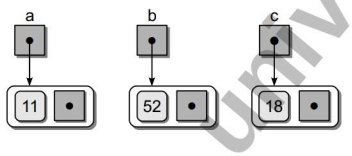
Kita mulai dengan membuat class yang akan menjadi objek-objek yang saling dihubungkan itu.

```
1 class Node(object):
2     """Sebuah simpul di linked list"""
3     def __init__(self, data, next=None):
4         self.data = data
5         self.next = next
```

Larikan, atau import, programnya. Lalu jalankan

```
a = Node(11)
b = Node(52)
c = Node(18)
```

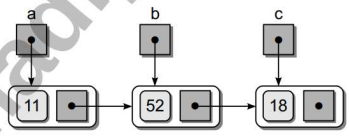
Hasilnya adalah tiga variabel dengan tiga objek[6]:



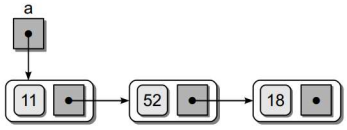
Mari kita sambungkan:

```
a.next = b
b.next = c
```

yang lalu menghasilkan yang berikut:



Kita bisa melupakan[7] variabel `b` dan `c`, dan lalu menaruh perhatian pada struktur berikut:

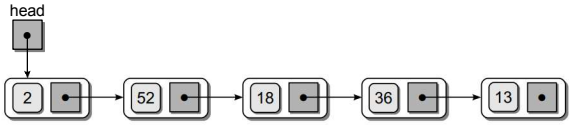


Kita bisa mengakses tiap-tiap simpul seperti ini (cobalah!):

```
print(a.data)
print(a.next.data)
print(a.next.next.data)
```

[6] Gambar-gambar linked list di sini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.

[7] Misal dengan menge-set masing-masing variabel itu ke `None`, seperti ini: `>>> b = None; c = None`



Gambar 3.2: Sebuah linked-list tertaut tunggal berisi lima simpul dan sebuah acuan ke *head*

Sebuah linked list umumnya diakes lewat ***head***-nya, seperti diperlihatkan di Gambar 3.2. Sebuah linked list bisa juga kosong, ditunjukkan dengan *head* yang berisi `None`. Simpul terakhir sebuah linked list disebut ***ekor*** atau ***tail***, yang mana tautnya berisi `None` (tidak menunjuk ke mana-mana). Setelah kita mempunyai sebuah objek linked list (yang diwakili oleh *head*-nya), kita bisa melakukan operasi-operasi seperti berikut

- mengunjungi dan mencetak data di tiap simpul
- mencari data yang isinya tertentu
- menambah suatu simpul di awal
- menambah suatu simpul di akhir
- menyisipkan suatu simpul di mana saja
- menghapus suatu simpul di awal, di akhir, atau di mana saja

Mengunjungi Setiap Simpul dari Depan

Ini cukup mudah. Ketiklah kode berikut di bawah class di atas lalu larikan

```
7  def kunjungi( head ):
8      curNode = head
9      while curNode is not None :
10         print(curNode.data)
11         curNode = curNode.next
```

Lalu buatlah sebuah struktur linked list seperti yang kamu lakukan sebelumnya. Lalu cobalah algoritma `kunjungi()` di atas dengan mengetik yang berikut di Python Shell:

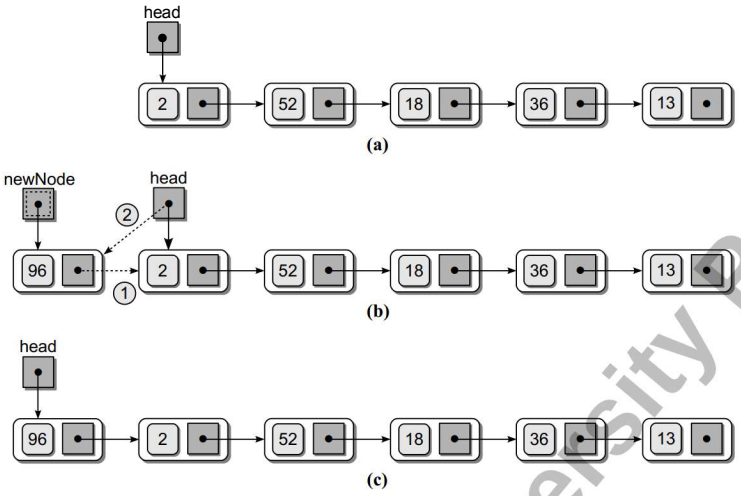
```
kunjungi(a)
```

Bagaimanakah hasilnya?

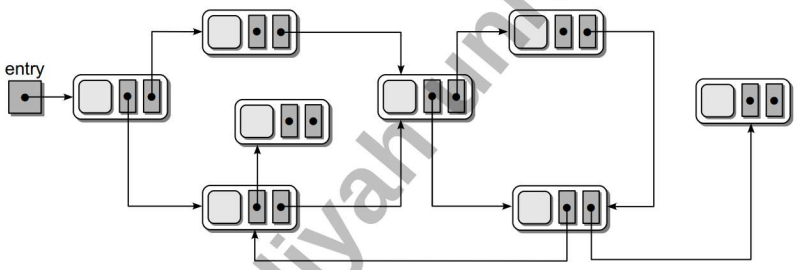
Menambah sebuah simpul di awal

Gambar 3.3 menunjukkan proses penambahan simpul di depan. Buatlah programnya!

Tugasmu sekarang adalah membuat kode untuk operasi-operasi yang lain (menyisipkan, menghapus, dll.) seperti yang telah diterangkan di kuliah.



Gambar 3.3: Ilustrasi penambahan simpul di depan



Gambar 3.4: Contoh sebuah *complex linked structure*.

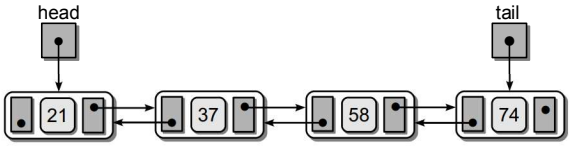
Advanced Linked List

Linked list yang dibahas di atas adalah bentuk dasar dan minimalnya. Terdapat banyak hal yang bisa ditingkatkan dan dimodifikasi untuk keperluan lain. Gambar 3.4 menunjukkan sebuah linked structure yang lebih kompleks.

Linked list bisa juga berupa *doubly linked list*, di mana tiap simpul bertautan dengan simpul lain secara dua arah. Gambar 3.5 menunjukkan ilustrasinya, dan program berikut membuat class-nya.

```
1 class DNode(object) :
2     def __init__( self, data ):
3         self.data = data
4         self.next = None
5         self.prev = None
```

Bisa kamu lihat bahwa penambatnya ada dua: `prev` dan `next`.



Gambar 3.5: *Double linked list.*

3.4 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, kerjakan dulu latihan-latihan di atas.

1. Terkait array dua dimensi, kita akan membuat tipe data sebuah matrix yang berisi angka-angka. Untuk itu buatlah fungsi-fungsi
   - untuk memastikan bahwa isi dan ukuran matrix-nya konsisten (karena tiap anggota dari list-luar-nya bisa saja mempunyai ukuran yang berbeda-beda, dan bahkan bisa saja berbeda tipe!),
   - untuk mengambil ukuran matrixnya,
   - untuk menjumlahkan dua matrix (pastikan ukurannya sesuai),
   - untuk mengalikan dua matrix (pastikan ukurannya sesuai),
   - untuk menghitung determinan sebuah matrix bujursangkar.
2. Terkait matrix dan *list comprehension*, buatlah (dengan memanfaatkan *list comprehension*) fungsi-fungsi
   - untuk membangkitkan matrix berisi nol semua, dengan diberikan ukurannya. Pemanggilan: `buatNol(m,n)` dan `buatNol(m)`. Pemanggilan dengan cara terakhir akan memberikan matrix bujursangkar ukuran $m \times m$.
   - untuk membangkitkan matrix identitas, dengan diberikan ukurannya. Pemanggilan: `buatIdentitas(m)`.
3. Terkait linked list, buatlah fungsi untuk
   - mencari data yang isinya tertentu: `cari(head,yang_dicari)`
   - menambah suatu simpul di awal: `tambahDepan(head)`
   - menambah suatu simpul di akhir: `tambahAkhir(head)`
   - menyisipkan suatu simpul di mana saja: `tambah(head,posisi)`
   - menghapus suatu simpul di awal, di akhir, atau di mana saja: `hapus(posisi)`
4. Terkait *doubly linked list*, buatlah fungsi untuk
   - mengunjungi dan mencetak data tiap simpul *dari depan* dan *dari belakang*.
   - menambah suatu simpul di awal

- menambah suatu simpul di akhir