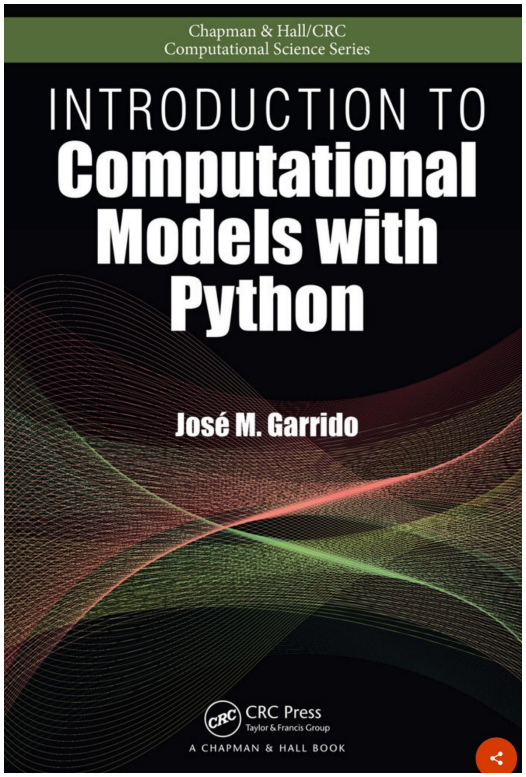

INTRODUCTION TO COMPUTATIONAL MODELS WITH PHYTON

Tugas Mata Kuliah SK5003 Pemrograman dalam Sains
Summary Chapter VII

Mohammad Rizka Fadhli (Ikang)
20921004@mahasiswa.itb.ac.id

18 September 2021

Contents

List of Figures

1 CHAPTER VII

Lists, Strings, and Sequences

1.1 *Lists*

Di beberapa *programming language*, kita mengenal tipe data berbentuk *vector* atau *arrays*, yakni sebuah himpunan atau barisan dari beberapa data (*entry*).

Di *Python*, kita menggunakan istilah *lists* karena merupakan bentuk yang lebih umum untuk merepresentasikan *array*.

> *List* adalah sekumpulan *ordered items* yang bisa memiliki berbagai macam tipe data.

Untuk melakukan definisi dari *lists* di Python, kita cukup menggunakan perintah:

```
v = [p1, p2, p3, ..., pn]
```

Kita menggunakan *bracket*, yakni `[ ]` berisi *items* yang hendak kita masukkan.

Sebagai contoh, saya akan membuat satu *array* yang saya beri nama `bilangan`.

```
>>> bilangan = [1,2,3,4,5,6,7,8,9,10]
```

1.1.1 *Index List*

Python memiliki sistem *indexing* dimulai dari `0`. Hal ini berbeda dengan *programming languages* lain yang memulai *indexing* dari `1`.

Perhatikan `bilangan` yang telah kita definisikan sebelumnya:

```
>>> bilangan
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Jika kita hendak mengambil elemen pertama dari `bilangan`, kita perlu menambahkan *index* `[0]` setelah nama *object*-nya.

```
>>> bilangan[0]
1
```

Demikian pula jika hendak mengambil elemen kedua dari `bilangan`, kita perlu menambahkan index `[1]` setelah nama *object*.

```
>>> bilangan[1]
2
```

Dari sini kita bisa mendapatkan bentuk umum sebagai berikut:

> Untuk data berbentuk *list* di Python, elemen ke n dipanggil dengan cara menuliskan $list[n-1]$.

Mari kita lihat lagi contohnya sebagai berikut:

```
>>> bilangan[9]
10

>>> n = 5
>>> bilangan[n]
6
```

1.1.1.1 *Index* Negatif Python bisa mendapatkan input *index* negatif. Jika kita perhatikan kembali, `bilangan` memiliki 10 elemen (*index* `[0-9]`). Kita bisa memberikan input *index* `[-1]` yang berarti **elemen pertama dari akhir**.

```
>>> bilangan[-1]
10
```

Demikian juga dengan nilai *index* negatif lainnya. Mari kita coba:

```
>>> bilangan[-2]
9

>>> bilangan[-3]
8

>>> bilangan[-4]
7
```

1.1.2 Mengubah Nilai dalam *List*

Dari *list* bernama `bilangan` di atas, kita bisa mengubah nilai elemen yang ada di sana dengan cara memanggil elemennya dan melakukan pendefinisian ulang.

Misalkan, kita hendak mengambil elemen ke-`5` dan mengubahnya menjadi bilangan lain. Misalkan $bilgangan_5 = 101$. Maka berikut adalah perintahnya:

```
>>> bilangan[4] = 101
```

Mari kita lihat hasilnya:

```
>>> bilangan
[1, 2, 3, 5, 101, 7, 8, 9, 10]
```

1.1.3 *List Slicing*

Ada kalanya kita hanya ingin bekerja di beberapa elemen dalam suatu *list* saja. Kita tidak memerlukan keseluruhan elemen *list* tersebut. Maka kita bisa memilih elemen-elemen yang diinginkan dari *list* tersebut.

Proses tersebut bernama *slicing operation*.

Sebagai contoh, kita hanya ingin mengeluarkan $bilangan_1, bilangan_2, bilangan_3$. Maka yang perlu diketikkan ke dalam Python adalah:

```
>>> bilangan
[1, 2, 3, 4, 101, 6, 7, 8, 9, 10]

>>> bilangan[0:3]
[1, 2, 3]
```

Selain melakukan *slicing* seperti di atas, kita bisa menyimpan hasil *slicing* ke dalam variabel baru (*assigned as new variable*). Sebagai contoh, saya akan membuat *list* baru bernama `number` yakni berisi $bilangan_i, i \in [5, 10]$

```
>>> number = bilangan [4:9]
>>> number
[101, 6, 7, 8, 9]
```

Slicing bisa juga dilakukan untuk melakukan *assignment* terhadap *slicing* yang lain. Berikut contohnys:

Kita hendak mengubah $bilangan_1 \rightarrow bilangan_2$ dan $bilangan_2 \rightarrow bilangan_3$.

```
>>> bilangan[0:2] = bilangan[1:3]
>>> bilangan
[2, 3, 3, 4, 101, 6, 7, 8, 9, 10]
```

Cara lain melakukan *slicing* adalah dengan menggunakan perintah `:` dalam *bracket* `[ ]`. Agar memudahkan, saya akan *define list* baru bernama `bil_baru` berisi `[0,1,2,3,4,5,6]`.

```
>>> bil_baru = [0,1,2,3,4,5,6]
>>> bil_baru
[0, 1, 2, 3, 4, 5, 6]
```

Misalkan saya hendak mengambil elemen `bil_baru` dengan *index* `1` ke atas:

```
>>> bil_baru[1:]
[1, 2, 3, 4, 5, 6]
```

Jika ingin mengambil elemen `bil_baru` dengan *index* `4` ke atas:

```
>>> bil_baru[4:]
[4, 5, 6]
```

Sedangkan jika ingin mengambil elemen `bil_baru` dengan *index* `5` ke bawah:

```
>>> bil_baru[:5]
[0, 1, 2, 3, 4]
```

Jika ingin mengambil elemen `bil_baru` dengan *index* `3` ke bawah:

```
>>> bil_baru[:3]
[0, 1, 2]
```

1.1.4 Fungsi `len()`

Fungsi `len()` digunakan untuk mengukur seberapa banyak isi elemen dari suatu *list*. Contohnya:

```
>>> bilangan
[2, 3, 3, 4, 101, 6, 7, 8, 9, 10]

>>> len(bilangan)
10

>>> bil_baru
[0, 1, 2, 3, 4, 5, 6]

>>> len(bil_baru)
7
```

1.1.5 Fungsi `range()`

Fungsi `range()` digunakan untuk membuat suatu *sequences*. Biasanya kita gunakan `range()` dalam *looping*. Contoh:

```
>>> for i in range(5):
      print(i)
0
1
2
3
4
```

1.2 Iterasi dalam Suatu *List*

Fungsi `len()` dari suatu *list* bisa digunakan sebagai bagian dari iterasi tertentu. Contohnya adalah misalkan saya memiliki data `20` berat badan sebagai berikut:

```
>>> berat_badan = [34, 44, 48, 41, 14, 15, 13, 19, 8, 38,
                   16, 43, 39, 32, 5, 18, 49, 29, 2, 35]
```

Misalkan saya hendak menghitung menggunakan *looping*, saat `berat_badan` ≥ 30, kita akan `print()` *index* dari elemennya dan kita akan hitung rata-ratanya dengan cara:

$$\bar{bb} = \frac{\sum bb}{n}$$

Berikut adalah perintah dan hasilnya:

```
>>> n_bb = range(len(berat_badan))
>>> total = 0.0
>>> n = 0
>>> for i in n_bb :
      if berat_badan[i] >= 30 :
          print("Index dg bb >= 30 ada di: ",i)
          total = total + berat_badan[i]
          n += 1
Index dg bb >= 30 ada di:  0
Index dg bb >= 30 ada di:  1
Index dg bb >= 30 ada di:  2
Index dg bb >= 30 ada di:  3
Index dg bb >= 30 ada di:  9
Index dg bb >= 30 ada di:  11
```

```
Index dg bb >= 30 ada di:  12
Index dg bb >= 30 ada di:  13
Index dg bb >= 30 ada di:  16
Index dg bb >= 30 ada di:  19

>>> print("Rata-rata: ",total/n)
Rata-rata:  40.3
```

1.3 Membuat *List* Menggunakan *Loop*

Suatu *list* bisa dibuat dari *list* yang kosong terlebih dahulu. Sebagai contoh, kita membuat satu *list* kosong bernama `listk` dan direncanakan akan memiliki elemen sebanyak `10` sebagai berikut:

```
>>> listk = []
>>> n_el = 10
```

Kita akan isi `listk` dengan angka-angka kelipatan `7`. Berikut skripnya:

```
>>> for j in range(1, n_el+1):
      item = j * 7
      listk.append(item)
>>> print("Hasilnya sebagai berikut: ", listk)
Hasilnya sebagai berikut:  [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
```

1.4 Membuat *Function* untuk Membuat *List*

Dari bagian sebelumnya, kita bisa mendefinisikan *function* untuk membuat suatu *list* berisi `n` elemen dan kelipatan bilangan `k` tertentu.

Misalkan:

```
>>> def generate (nama_list, n_elemen, k_kelipatan) :
      for i in range(1, n_elemen+1) :
        item = i * k_kelipatan
        nama_list.append(item)
    print("Berikut adalah hasilnya: ",nama_list)
```

Jika saya *run* dengan beberapa kondisi, didapatkan hasil berikut:

```
>>> list_5 = []
>>> n_5 = 5
>>> k_5 = 5
>>> generate(list_5, n_5, k_5)
Berikut adalah hasilnya:  [5, 10, 15, 20, 25]
```

Mari kita coba untuk *input* lainnya:

```
>>> list_2 = []
>>> n_2 = 15
>>> k_2 = 2
>>> generate(list_2, n_2, k_2)
Berikut adalah hasilnya:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
```

1.5 Menggabungkan Beberapa *Lists* dengan `extend()`

Kita bisa menggambungkan beberapa *lists* dengan cara melakukan *extending*. Misalkan ada dua *lists* sebagai berikut:

```
>>> bil_1 = [2, 4, 8, 10, 12, 14]
>>> bil_2 = [3, 7, 9]
```

Kita akan masukkan semua elemen `bil_2` ke dalam `bil_1` dengan cara:

```
>>> bil_1.extend(bil_2)
>>> bil_1
[2, 4, 8, 10, 12, 14, 3, 7, 9]
```

1.6 *Insert* Elemen ke Dalam *List*

Kita bisa memasukkan (*insert*) suatu elemen ke *list* di *index* tertentu dengan perintah `nama_list.insert(index, nilai)`.

```
>>> bil_1.insert(2,100)
>>> bil_1
[2, 4, 100, 8, 10, 12, 14, 3, 7, 9]
```

1.7 Menghapus Elemen *List*

Selain menambahkan, kita juga bisa menghapus elemen suatu *list*. Sebagai contoh, saya memiliki *list* bernama `prime` berikut:

```
>>> prime = [2,3,5,7,11,13,17,23,11,29]
>>> prime
[2, 3, 5, 7, 11, 13, 17, 23, 11, 29]
```

Jika kita hendak menghapus elemen dengan nilai `11`, maka kita lakukan:

```
>>> prime.remove(11)
>>> prime
[2, 3, 5, 7, 13, 17, 23, 11, 29]
```

Ternyata Python hanya menghapus elemen `11` yang pertama saja. Ketika kita ingin menghapus elemen `11` yang lainnya, kita cukup ulang:

```
>>> prime.remove(11)
>>> prime
[2, 3, 5, 7, 13, 17, 23, 29]
```

Jika kita hendak menghapus elemen terakhir, kita perlu lakukan:

```
>>> prime.pop()
29

>>> prime
[2, 3, 5, 7, 13, 17, 23]
```

`prime.pop()` bertugas untuk **menampilkan dan menghapus** elemen terakhir dari *list* bernama `prime`.

1.8 Menemukan *Index* dari Suatu Elemen *List*

Adakalanya kita hendak mencari dimana letak elemen tertentu dari suatu *list*. Misalkan kita hendak mencari di elemen ke berapa angka `13` muncul. Kita bisa gunakan perintah berikut:

```
>>> prime.index(13)
4
```

Terlihat bahwa angka `13` berada di elemen ke `4`.

2 Hubungan *Function* dan *List*

Kita bisa membuat suatu *function* yang memiliki *input* sebuah *list* dan mengeluarkan *output* berupa *list* kembali. Sebagai contoh, saya hendak membuat *function* untuk menghitung BMI dari *input* berupa dua buah *lists* `berat_badan` dan `tinggi_badan`.

```
>>> def hitung_bmi (berat_badan,tinggi_badan) :
      n = len(bb)
      bmi = []
      for i in range(0,n):
        bmi_hit = berat_badan[i] / (tinggi_badan[i]/100)**2
        bmi_hit = round(bmi_hit,2)
        bmi.append(bmi_hit)
      print(bmi)

>>> bb = [77,78,60,56,83,50,63,74,62,70]
>>> tb = [152,181,159,157,185,172,190,173,150,154]

>>> hitung_bmi(bb,tb)
[33.33, 23.81, 23.73, 22.72, 24.25, 16.9, 17.45, 24.73, 27.56, 29.52]
```

3 *Generating Sequences* ke dalam *List*

Ada kalanya kita perlu membuat *sequence* dan dimasukkan ke dalam *list*.

3.1 *Sequence* Repetisi

Misal, kita hendak membuat suatu *list* dengan `8` elemen berisi nilai `5`. Berikut adalah caranya:

```
>>> n = 8
>>> seq = [5 for i in range(n)]
>>> seq
[5, 5, 5, 5, 5, 5, 5, 5]
```

Selain cara tersebut, kita juga bisa memanfaatkan perintah `append()` dan *loop* `for` seperti ini:

```
>>> seq_2 = []
>>> for i in range(n):
     seq_2.append(5)

>>> seq_2
[5, 5, 5, 5, 5, 5, 5, 5]
```

Kita bisa memodifikasi kalkulasi yang ada dengan berbagai fungsi agar tercipta suatu deret dengan nilai yang berbeda-beda.

Contoh:

```
>>> n = 10
>>> seq_3 = [i + 2 for i in range(n)]
>>> seq_3
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

4 *List* di Dalam *List*

Suatu bentuk data berupa *list* di dalam *list* disebut dengan *nested list* atau *multidimensional arrays*. Misalkan kita akan membuat satu *list* bernama `daftar_1`, kemudian akan kita masukkan ke dalam *list* berikutnya bernama `daftar_2`.

```
>>> daftar_1 = [2,4,3]
>>> daftar_2 = [9,7,5,daftar_1,10,4]
>>> daftar_2
[9, 7, 5, [2, 4, 3], 10, 4]
```

Kita telah lihat bahwa pada *index* ke `3` bentuknya adalah *list*. Jika kita hitung berapa elemen, didapatkan:

```
>>> len(daftar_2)
6
```

Tetap `6` elemen. *List* di elemen ke `3` tetap dianggap menjadi 1 elemen. Mari kita coba panggil `daftar_1` dari `daftar_2`:

```
>>> daftar_2[3]
[2, 4, 3]
```

Untuk memanggil elemen pertama dari `daftar_1` melalui `daftar_2` adalah sebagai berikut:

```
>>> daftar_2[3][0]
2
```

5 Matriks dalam Python

Kita bisa membuat matriks dalam Python. Misalkan kita hendak mendefinisikan sebuah matriks berukuran 2 baris x 3 kolom berikut:

```
>>> smx = [[9,1,7],[2,6,1]]
>>> smx
[[9, 1, 7], [2, 6, 1]]
```

Seandainya kita hendak mengambil elemen matriks di baris pertama kolom pertama, maka:

```
>>> smx[0][0]
9
```

Jika hendak mengambil elemen matriks baris kedua kolom terakhir, maka:

```
>>> smx[1][2]
1
```

5.0.1 Membuat Matriks dengan Looping

Misalkan kita hendak membuat matriks berukuran `3 x 4` dengan isi berupa angka dari `1` hingga `12`. Kita bisa membuat *looping* sebagai berikut:

```
# membuat template matriks
matriks = [[]]
# initial
x = 1
# looping
for i in range(3):
  baris = []
  for j in range(4):
    baris.append(x)
    x += 1
  matriks.append(baris)

# oleh karena elemen pertama merupakan baris = []
# kita perlu menghapusnya dengan cara
matriks.pop(0)

# menampilkan output
print(matriks)
```

Saya *save* program tersebut bernama `matriks.py` dan saya *run*, sehingga menghasilkan:

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Selain menggunakan *looping* di atas, kita bisa menggunakan perintah *nested list* sebagai berikut:

```
>>> matriks_2 = [[1 for i in range(4)] for j in range (3)]
>>> matriks_2
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
```

6 *Tuples*

Tuples adalah bentuk *sequence* dalam Python selain *list*. Perbedaannya adalah Tuples *immutable*. Isinya tidak bisa diubah dan elemennya tidak bisa ditambah. Namun kita tetap bisa membuat *nested tuples*.

Contoh:

```
>>> tp = (2,5,8,3,9,4)
>>> tp
(2, 5, 8, 3, 9, 4)
```

Kita tetap bisa memanggil elemen dengan *index*.

```
>>> tp[2]
8
```

Berikut adalah contoh dari *nested tuples*:

```
>>> tp2 = (1,2,tp)
>>> tp2
(1, 2, (2, 5, 8, 3, 9, 4))
```

Menghitung panjang *tuples* dengan`len()`:

```
>>> len(tp2)
3
```

6.1 *Converting Tuples* ke dalam *List*

Kita bisa ubah suatu *tuples* menjadi *list* dengan cara:

```
>>> tp_new = list(tp)
>>> tp_new
[2, 5, 8, 3, 9, 4]
```

6.2 *Converting List* ke dalam *Tuples*

Kita juga bisa mengembalikan prosesnya sebagai berikut:

```
>>> tp_new_2 = tuple(tp_new)
>>> tp_new_2
(2, 5, 8, 3, 9, 4)
```

6.3 *Tuples* di dalam *List*

Kita bisa membuat *list* berisi *tuples*:

```
>>> new = [(2,4,3),(1,9,6),(3,8,6)]
>>> new
[(2, 4, 3), (1, 9, 6), (3, 8, 6)]
```

6.4 *List* di dalam *Tuples*

Atau kita bisa membuat kebalikannya:

```
>>> baru = ([6,5,3],[4,5,9],[9,4,6])
>>> baru
([6,5,3],[4,5,9],[9,4,6])
```

7 *Dictionaries*

Ada kalanya kita membutuhkan struktur data yang memiliki *index* bukan angka, tapi suatu *key*. Kita bisa menggunakan bentuk *dictionaries* berikut:

```
>>> dct = {'nama' : 'ikang', 'nim' : 20921004, 'tinggi' : 175}
>>> dct
{'nama': 'ikang', 'nim': 20921004, 'tinggi': 175}
```

Kita bisa memanggil data yang ada dengan menggunakan *key index* berikut:

```
>>> dct['nama']
'ikang'
```

Kita bisa mengubah isi *dictionaries* dengan cara mendefinisikan ulang via *index key*:

```
>>> dct['nama'] = "MRF"
>>> dct
{'nama': 'MRF', 'nim': 20921004, 'tinggi': 175}
```

8 *Strings*

Strings adalah data dalam bentuk karakter.

Contoh:

```
>>> pesan = 'selamat pagi'
>>> pesan
'selamat pagi'
```

8.1 *Concatenante*

Yakni menempel *string* lain ke *string* yang sudah ada. Contoh:

```
>>> pesan_2 = pesan + ' ' + dct['nama']
>>> pesan_2
'selamat pagi MRF'
```

8.2 Beberapa Perintah Lain Terkait *String*

```
>>> len(pesan_2) # panjang string
16

>>> pesan_2[10]
'g'

>>> print(pesan_2[1:4])
ela

>>> print(pesan_2[0:8])
selamat

>>> "sela" in pesan_2
True

>>> pesan_3 = pesan_2 + "\nHave a nice day"
>>> pesan_3
'selamat pagi MRF\nHave a nice day'

>>> print(pesan_3)
selamat pagi MRF
Have a nice day
```

```
>>> pesan_4 = pesan_2 + "\nhave a \"nice\" day"
>>> pesan_4
'selamat pagi MRF\nhave a "nice" day'

>>> print(pesan_4)
selamat pagi MRF
have a "nice" day

>>> pesan_4.find("day")
31

>>> number = "0810"
>>> number.isdigit()
True
```

== End ==