PROGRAM STUDI INFORMATIKA
FAKULTAS KOMUNIKASI
DAN INFORMATIKA

Modul Praktikum

# Algoritma & Struktur Data

Versi 4.3

Fajar Suryawan, PhD
Husni Thamrin, PhD
Wiwit Supriyanti, S.Kom.,M.Kom
Dimas Aryo Anggoro, S.Kom.,M.Sc
Yunita Ardiyanto



# Modul Praktikum
# Algoritma & Struktur Data

*Versi 4.3*

Fajar Suryawan, PhD
Husni Thamrin, PhD
Wiwit Supriyanti, S.Kom.,M.Kom
Dimas Aryo Anggoro, S.Kom.,M.Sc
Yunita Ardiyanto

# Modul Praktikum
# **Algoritma & Struktur Data**

*Versi 4.3*

***Penulis:***
Fajar Suryawan, PhD
Husni Thamrin, PhD
Wiwit Supriyanti, S.Kom.,M.Kom
Dimas Aryo Anggoro, S.Kom.,M.Sc
Yunita Ardiyanto

***Layouter***
Fajar Suryawan

***Desain Cover:***
Ali Himawan

**ISBN:** 978-602-361-279-6
Cetakan 1, Januari 2020



Penerbit
Muhammadiyah University Press
Universitas Muhammadiyah Surakarta
Gedung I Lantai 1
Jl. A Yani Tromol Pos 1 Pabelan Kartasura Surakarta 57102
Jawa Tengah - Indonesia
Telp : (0271) 717417 Ext. 2172
Email : muppress@ums.ac.id

# Pengantar

Alhamdulillah, segala puji bagi Allah subhanahu wa ta'ala yang telah memberi hidayah dan kesempatan sehingga penulis dapat menyelesaikan modul praktikum Algoritma dan Struktur Data ini.

Seperti halnya matakuliah Algoritma dan Pemorgraman semester lalu, materi matakuliah ini juga mengalami perubahan yang signifikan dibandingkan dengan materi tahun-tahun sebelumnya. Kalau beberapa tahun yang lalu matakuliah ini memakai bahasa C sebagai kendaraannya, sekarang kita memakai bahasa Python.

Python adalah bahasa baru yang popularitasnya terus naik. Python dikenal karena, di antaranya, kesederhanaan syntax-nya dan modularitasnya. Semoga ini mempermudah para mahasiswa dalam mempelajari konsep Algoritma dan Struktur Data.

Pada edisi kedua (tahun 2015), panduan praktikum ini mempunyai tambahan sebuah modul baru, yakni *Regular Expressions*.

Pada edisi ketiga (tahun 2016), panduan praktikum ini mempunyai tambahan sebuah modul baru, yakni *Pohon Biner*. Sesuai dengan kurikulum baru tahun 2016, matakuliah ini sekarang diselenggarakan di Semester 4.

Pada edisi keempat beta-1 (Februari tahun 2019), panduan praktikum ini berpindah dari Python 2 ke Python 3. *Typos* yang ditemukan telah diperbaiki. Sejumlah soal baru diberikan. Terdapat pula penggabungan modul dan modul baru.

Pada edisi 4.3 (Mei 2019), panduan praktikum ini mempuyai modul baru, yakni Modul 10 *Analisis Algoritma*.

Selamat belajar!

Surakarta, 18 Mei 2019

Fajar Suryawan, PhD

# Daftar Isi

# Modul 1

# Tinjauan Ulang Python

Di bab ini akan kita tinjau ulang bahasa Python yang sudah dipelajari di kuliah semester lalu. **Pembahasan di bab ini tidak akan komprehensif dan tidak bisa menggantikan buku Python yang bagus ataupun kuliah pemrograman satu semester**. Dengan demikian mahasiswa diminta mereview materi kuliah semester lalu secara menyeluruh dan menyiapkan buku acuan yang memadai untuk mendampingi pemakaian Python pada beberapa bulan ke depan.

**Penting:** mahasiswa diminta mengerjakan soal-soal latihan di modul praktikum ini dengan sungguh-sungguh. Ingat! Apa yang akan kamu panen besok bergantung dari apa yang kamu tanam hari ini.

Pastikan bahwa program Python versi 3.7.2 sudah terinstal di komputermu. Kalau belum, silakan unduh dari `python.org`. Disarankan pula untuk mengunduh IDLEX atau Thonny sebagai IDE-nya. Kamu dapat mengunduhnya dengan mengetik `pip install idlex` atau `pip install thonny` di *command prompt*.

## 1.1 Memulai Python

Nyalakan Python. Kamu akan dihadapkan pada jendela Python dengan prompt yang siap diisi seperti ini:

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52)
[MSC v.1916 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Kita mulai dengan menggunakan Python sebagai kalkulator. Pada prompt, ketikkan yang berikut ini diikuti menekan tombol <Enter>:

```
>>> 2 + 3
```

Apa yang kamu dapatkan? Lanjutkan:

```
>>> 2+3*5-6/2
```

Perhatikan bahwa perkalian dan pembagian lebih didahulukan ketimbang penjumlahan dan pengurangan. Kamu dapat mengubah *precedence* ini dengan memakai tanda kurung. Python juga dapat dipakai untuk menghitung sampai bilangan yang besar. Berapakah $2^{1000}$? Ketik ini:

```
>>> # Menghitung dua pangkat seribu
>>> 2**1000
```

Simbol # menandakan komentar dan baris itu tidak akan dieksekusi oleh Python interpreter.

### Variabel dengan tipe data `int` dan `float`

Variabel adalah memberi nama pada suatu objek. Coba yang berikut ini:

```
>>> radius = 4
>>> pi = 3.14159
>>> area = pi * radius * radius
>>> print(area)
```

Jika kita memberi nilai pada variabel yang sudah ada, nilainya akan berganti:

```
>>> x = 4
>>> print(x)
4
>>> x = 5
>>> print(x)
5
```

Pada kode-kode di atas kita telah berkenalan dengan tipe data `int` dan `float`. Kamu bisa memeriksa tipe data suatu variable dengan fungsi `type()`. Contoh:

```
>>> a = 5
>>> b = 6.2
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
```

### String

String adalah serangkaian karakter. Beri nilai 'Apa kabar' pada string s seperti berikut[1]:

```
>>> s = 'Apa kabar'
>>> s
'Apa kabar'
>>> type(s)
<class 'str'>
```

Tipe data variable `s` adalah `str` (maksudnya 'string'). Karakter juga dapat 'dijumlahkan'. Coba ini:

```
>>> a = 'Halo'
>>> b = ' mas'
>>> c = ' Data!'
```

[1] Gunakan tanda petik tunggal atau petik ganda, yang letak kuncinya di keyboard-mu adalah di sebelah kiri tombol <Enter>. JANGAN pakai tanda petik yang terletak di bagian kiri atas keyboard sebelah angka 1.

```
>>> d = a + b + c
>>> d
'Halo mas Data!'
```

Kamu tidak bisa menjumlahkan variabel dengan tipe data `'str'` dengan variable dengan tipe data `'int'`. Namun kamu dapat merubah, jika keadaannya tepat, tipe data suatu variabel. Seperti ini:

```
>>> g = '34'
>>> h = 23
>>> g + h
Traceback (most recent call last):
File "<pyshell#7>", line 1, in <module>
g+h
TypeError: can only concatenate str (not "int") to str
>>> int(g) + h
57
```

Variable `g` adalah bertipe data `str` (karena ada tanda kutipnya). Jadi tidak bisa dijumlahkan dengan `h`, yang bertipe data `int`. Tapi perintah `int(g)` membuat sebuah objek baru yang bertipe data integer (bilangan bulat).

## 1.2 List dan Tuple (dan String lagi)

List adalah sekumpulan objek yang berurutan. Tuple sama seperti itu, hanya saja dia tidak bisa diubah nilai elemennya. Perhatikan yang berikut ini:

```
>>> f = 2.5
>>> g = 7
>>> d = [f, g, 3.9, 8, 'Apa kabar']
>>> d
[2.5, 7, 3.9, 8, 'Apa kabar']
>>> type(d)
<class 'list'>
```

Variabel `d` adalah sebuah list dengan elemen-elemen seperti ditampilkan di atas. Kita bisa mengakses dan mengubah tiap-tiap elemennya seperti ini

```
>>> d[0] = 55
>>> d
[55, 7, 3.9, 8, 'Apa kabar']
```

Tampak bahwa elemen ke-0 sudah berubah nilainya. Kamu juga dapat mengakses elemen-elemen yang lain dengan index-nya. Ingat pelajaran tentang slicing dan index negatif.

Suatu list dapat di-*iterate* ("ditunjuk satu-per-satu secara urut") seperti ini:

```
>>> for i in d:
        print(i)   ## di sini pencet '<Enter>' dua kali


55
7
3.9
8
Apa kabar
>>>
```

Sesungguhnya semua objek yang bersifat *iterable* dapat diiterasi seperti di atas, dapat 'diiris' (*sliced*, lihat di bawah), dan 'dapat dikenai pencarian'.

### Slicing

Slicing adalah mengiris elemen-elemen di suatu list atau tuple dengan pola tertentu. Ketik seperti berikut

```
>>> a = 'Wacana keilmuan dan keislaman'
>>> b = [43,44,45,46,47,48,49,50]
```

Sekarang cobalah yang berikut ini dan amati hasilnya

```
a[0:6]
a[7:15]
a[::-1]
a[-7:-2]
a[-7:100]
len(a)
a[0:29]
a[0:100]
a[0:29:2]
a[0:200:2]
```

Lakukan hal yang serupa untuk variable `b` di atas.

## 1.3 Dictionary

Dictionary (atau `dict`) adalah tipe data di Python yang sangat berguna. Akan kita bahas secara ringkas.

Kalau list atau tuple mempunyai index bilangan bulat (saja), maka dict mempunyai index (disebut 'key') yang bisa berupa tipe data lainnya. Dictionary menyimpan data dalam bentuk **pasangan kunci–nilai**. Berikut ini contohnya

```
>>> dd = {'nama':'joko', 'umur':21, 'asal':'surakarta'}
>>> dd['nama']
'joko'
>>>
```

Untuk membuat sebuah dictionary yang kosong, gunakan perintah `d = dict()` atau `d = {}`.

## 1.4 Set

Set atau himpunan adalah struktur data yang mirip sebuah list, tapi elemennya tidak berulang. Untuk membuat sebuah set, bisa kita gunakan kurung kurawal ataupun fungsi `set()`. Untuk membuat set kosong, pakailah perintah `set()`, bukan {}. Yang terakhir ini adalah untuk dictionary kosong. Perlu diperhatikan bahwa anggota dalam `set` *tidak* mengenal urutan (beda dengan `list`, yang mengenal urutan). Jadi ketika suatu object `set` di-`print`, bisa jadi urutannya berubah.

Berikut ini adalah demonstrasi set. Silakan dijalankan!

```
>>> keranjang = {'apel', 'jeruk', 'apel', 'manggis', 'jeruk', 'pisang'}
>>> print(keranjang)    # tunjukkan bahwa duplikat telah dibuang
{'manggis', 'pisang', 'jeruk', 'apel'}
>>> 'jeruk' in keranjang
True
>>> 'rumput' in keranjang
False
>>>  ## Akan didemokan operasi set pada huruf unik di dua kata
>>> a = set('surakarta')
>>> b = set('yogyakarta')
>>> a                                        # huruf di a, tanpa diulang
{'s', 'u', 'k', 'a', 'r', 't'}
>>> b                                        # huruf di b, tanpa diulang
{'o', 'g', 'y', 'k', 'a', 'r', 't'}
>>> a - b                                    # huruf di a tapi tidak di b
{'s', 'u'}
>>> a | b                                    # huruf di a atau di b atau di keduanya
{'o', 'g', 'y', 's', 'u', 'k', 'a', 'r', 't'}
>>> a & b                                    # huruf di a yang sekaligus juga di b
{'r', 'k', 't', 'a'}
>>> a ^ b                          # huruf di a atau b tapi tidak sekaligus di keduanya
{'o', 'y', 'g', 's', 'u'}
>>>
```

## 1.5 Operator relasional dan tipe data Boolean

Tipe data boolean adalah tipe data dengan dua kemungkinan nilai: `True` ("benar")atau `False` ("salah"). Operator relasional adalah operator yang membandingkan dua variabel. Ketik contoh berikut:

```
p = 3
q = 7
p > q
p < q
p == q   # perhatikan bahwa ada dua tanda '='.
```

Kita juga dapat secara langsung mengetikkan langsung objeknya. Coba yang berikut:

```
4 < 8
4 > 8
4 == 4    # ada dua tanda '='.
4 < 4
4 <= 4
```

String juga dapat dibandingkan. Silakan ketik dan amati ini:

```
'UMS' > 'UGM'   ## True atau False? Jangan lupa memberi tanda kutipnya.
'UMS' > 'ITB'
'Emas' < 'Sayur'
'a' > 'b'
'a' < 'z'
'A' > 'a'
```

Tebak: bagaimanakah kira-kira antar string dibandingkan satu sama lain? Tipe data boolean ini dapat disimpan juga di suatu variabel. Silakan coba yang berikut:

```
v = 5 < 7      # v berisi True
ff = 'UMS' > 'UGM'
type(ff)
ff             # ketik nama variabelnya lalu tekan <Enter> .
g = 3 == 3
g
```

## 1.6 File .py

Perintah python dapat disimpan dalam suatu file .py dan dijalankan dengan menekan tombol F5 atau mengklik Run −> Run Module.

**Latihan 1.1** Buatlah suatu file baru lewat IDLE[2]. Simpan sebagai `LatReview1.py`. Ketikkan kode python berikut[3]

*LatReview1.py*

```
1 a = 4
2 b = 5
3 c = a + b
4 print('Nilai a =', a)
5 print('Nilai b =', b)
6 print('Sekarang, c = a + b =', a, '+', b, '=', c)
7 print('')
8 print('Sudah selesai.')
```

□

Larikan program di atas dengan memencet tombol F5 . Kamu akan mendapati ini di jendela Python Shell[4]:

```
Nilai a = 4
Nilai b = 5
Sekarang, c = a + b = 4 + 5 = 9

Sudah selesai.
```

[2]Klik 'File' lalu pilih 'New Window'. Atau pencet tombol Ctrl + N

[3]Aku ingatkan dirimu lagi, bahwa gunakanlah tanda petik tunggal atau petik ganda, yang letak kuncinya di keyboard-mu adalah di sebelah kiri tombol <Enter>. JANGAN pakai tanda petik yang terletak di bagian kiri atas keyboard sebelah angka 1.

[4]Jendela yang ada >>> -nya

### Mengambil nilai dari keyboard pengguna

Kita juga dapat mengambil nilai dari pengguna melalui keyboard. Perintah yang digunakan adalah

```
var = input('Silakan mengetik lalu tekan enter:> ')
```

Ketika ini dieksekusi, pengguna diminta memasukkan string lewat keyboard. Hasilnya (selalu bertipe string), ditampung dalam variable var.

**Latihan 1.2** Buatlah file berikut:

*LatReview2.py*

```
print('Kita perlu bicara sebentar...')
nm = input('Siapa namamu? (ketik di sini)> ')
print('Selamat belajar,', nm)
angkaStr = input('Masukkan sebuah angka antara 1 sampai 100 > ')
a = int(angkaStr)
kuadratnya = a*a
print(nm + ', tahukah kamu bahwa', a, 'kuadrat adalah', kuadratnya,'?')
```

Larikan program di atas dengan memencet tombol F5 . Jangan lupa untuk menjawab pertanyaan yang muncul! □

## 1.7 Fungsi

Fungsi adalah semacam prosedur yang bisa kita panggil sewaktu-waktu.

**Latihan 1.3** Kita mulai dengan membuat tiga buah fungsi dalam satu file.

*LatReview3.py*

```
def ucapkanSalam():
    print("Assalamu 'alaikum!")

def sapa(nama):
    ucapkanSalam()   # Ini memanggil fungsi ucapkanSalam() di atas.
    print('Halo',nama)
    print('Selamat belajar!')

def kuadratkan(b):
    h = b*b
    return h
```

Larikan program itu dengan menekan tombol F5 . Tidak terjadi apa-apa sepertinya... Tapi di memori sudah termuat fungsi-fungsi itu. Sekarang di Python Shell ketikkan perintah-perintah berikut[5]

```
ucapkanSalam()   # JANGAN TINGGALKAN tanda kurungnya!
sapa('Mas Wowok') # Atau, lebih baik lagi, cantumkan namamu.
b = kuadratkan(5)
b
k = 9
print('Bilangannya',k, ', kalau dipangkatkat dua jadinya', kuadratkan(k))
```

[5]Tentu saja, kalau pas di Python Shell komentarnya tidak perlu ditulis. Capek deh.

□

Perhatikan bahwa suatu fungsi bisa mempunyai ataupun tidak menerima parameter, bisa mengembalikan atau tidak mengembalikan sesuatu. Contoh fungsi yang menerima sesuatu dan mengembalikan sesuatu adalah fungsi `kuadratkan()` di atas.

Perhatikan juga bahwa Python memakai *indentation* (spasi ke kanan, biasanya sebanyak 4) untuk menandai blok eksekusi.

Beberapa bahasa lain –seperti C, C++, Java, C#, PHP, JavaScript– memakai kurung kurawal (simbol '{' dan '}') sebagai pembatas blok eksekusi. Bahasa yang lain lagi –seperti Pascal, Delphi, Lazarus, VHDL, LaTeX– memakai bentuk kata-kata `begin` — `end` untuk membuat blok eksekusi.

**Latihan 1.4** Buatlah sebuah fungsi `selesaikanABC()` yang menyelesaikan persamaan kuadrat

$$ax^2 + bx + c = 0. \tag{1.1}$$

Maksudnya, dengan diberi nilai $a, b, c$, cari (kalau ada) dua nilai $x_1$ dan $x_2$ yang memenuhi persamaan di atas. Ingat pelajaran SMA dulu bahwa solusi persamaan di atas diberikan oleh

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{1.2}$$

Ketik kode berikut

*LatReview4.py*

```
from math import sqrt as akar
def selesaikanABC(a,b,c):
    a =  float(a) # mengubah jenis integer menjadi float
    b =  float(b)
    c =  float(c)
    D = b**2 - 4*a*c
    x1 = (-b + akar(D))/(2*a)
    x2 = (-b - akar(D))/(2*a)
    hasil = (x1,x2)  # tuple yang terdiri dari dua elemen
    return hasil
```

Larikan program ini, lalu panggil di Python Shell seperti berikut

```
>>> k = selesaikanABC(1,-5,6)
>>> k
(3.0, 2.0)
>>> k[0]
3.0
>>> k[1]
2.0
```

Perhatikan bahwa kita memanggil fungsi dan mengembalikan hasilnya ke variabel `k`, yang bertipe data tuple.

Baris pertama file di atas: dari sebuah modul (`math`), meng-import sebuah fungsi (`sqrt`), lalu melekatkan ke sebuah nama '`akar`'.

Program di atas masih belum menangkap semua kemungkinan input. Jika dipanggil dengan `selesaikanABC(1,2,3)` akan menghasilkan sebuah error, karena determinan $D = b^2 - 4ac$ bernilai negatif.

Ubahlah program itu agar bisa menangkap kasus ini. □

## 1.8 Pengambilan keputusan

Kita mulai dengan beberapa contoh.

**Latihan 1.5** Buat suatu fungsi `apakahGenap()` yang mengembalikan objek boolean. Jika input ke fungsi itu genap, kembalikan `True`. Jika ganjil, kembalikan `False`.

File berikut bisa menjadi jawabannya (ketiklah!)

*LatReview5.py*

```
1 def apakahGenap(x):
2     if (x%2 == 0):
3         return True
4     else:
5         return False
```

Larikan kode di atas dengan memencet F5 . Di Python Shell, panggil fungsi itu:

```
apakahGenap(48)
apakahGenap(37)
```

Apakah hasilnya? □

**Latihan 1.6** Buat suatu fungsi yang apabila diberi suatu bilangan bulat positif akan mencetak suatu kalimat dengan aturan seperti berikut:

- Jika bilangan itu kelipatan 3, akan mencetak "Bilangan itu adalah kelipatan 3"
- Jika bilangan itu kelipatan 5, akan mencetak "Bilangan itu adalah kelipatan 5"
- Jika bilangan itu kelipatan 3 dan 5 sekaligus, akan mencetak "Bilangan itu adalah kelipatan 3 dan 5 sekaligus"
- Jika bilangan bukan kelipatan 3 ataupun 5, akan mencetak "Bilangan itu bukan kelipatan 3 maupun 5"

Yang di bawah ini bisa menjadi jawabannya (ketiklah):

*LatReview6.py*

```
def tigaAtauLima(x):
    if (x%3==0 and x%5==0):
        print('Bilangan itu adalah kelipatan 3 dan 5 sekaligus')
    elif x%3==0:
        print('Bilangan itu adalah kelipatan 3')
    elif x%5==0:
        print('Bilangan itu adalah kelipatan 5')
    else:
        print('Bilangan itu bukan kelipatan 3 maupun 5')
```

Larikan program itu dengan memencet tombol F5 , lalu panggil di Python Shell. Berikut ini contohnya.

```
>>> tigaAtauLima(9)
Bilangan itu adalah kelipatan 3
>>> tigaAtauLima(10)
Bilangan itu adalah kelipatan 5
>>> tigaAtauLima(15)
Bilangan itu adalah kelipatan 3 dan 5 sekaligus
>>> tigaAtauLima(17)
Bilangan itu bukan kelipatan 3 maupun 5
```

Renungkanlah jalan logika program di atas. □

**Latihan 1.7** Mencari seseorang di dictionary.

*LatReview7.py*

```
staff = { 'Santi'  : 'santi@ums.ac.id', \
          'Jokowi' : 'jokowi@solokab.go.id', \
          'Endang' : 'Endang@yahoo.com',\
          'Sulastri': 'Sulastri3@gmail.com' }

yangDicari = 'Santi'
if yangDicari in staff:
    print('emailnya', yangDicari, 'adalah' , staff[yangDicari])
else :
    print('Tidak ada yang namanya', yangDicari)
```

□

Bentuk pengambilan keputusan lain yang harus kamu ketahui adalah memakai `case` (tidak dibahas di sini).

## 1.9 Loop

Loop adalah perulangan. Meskipun semua perintah perulangan bisa dibuat dengan konstruk `if–elif–else`, banyak kemudahan yang bisa dicapai dengan memakai konstruk perulangan yang sudah tersedia. Ada dua konstruk yang akan kita bahas di sini: `for` dan `while`.

### Konstruk `For`

**Latihan 1.8** "Cetaklah bilangan dari 0 sampai 9." Ini dapat dicapai dengan program berikut

*LatReview8.py*

```
1 for i in range(0,10):
2     print(i)
```

yang menghasilkan, setelah menjalankan program di atas,

```
0
1
2
3
4
5
6
7
8
9
```

□

Seperti dijelaskan di halaman 4, semua objek yang *iterable* (`list`, `tuple`, `str`,...) dapat menjadi argumen untuk konstruk `for` ini.

**Latihan 1.9** Mencetak tiap-tiap huruf di sebuah kalimat. Misal kita punya string `s="Ini Budi"`, maka program berikut akan mencetak huruf-huruf di `s` satu persatu.

*LatReview9.py*

```
1 for i in s:
2     print(i)
```

Yang akan menghasilkan

```
I
n
i

B
u
d
i
```

□

Cobalah mengubah `s` di atas menjadi suatu list: `s=[4,3,2,5,6]`. Bagaimanakah hasilnya?

**Latihan 1.10** Mencetak isi suatu dictionary. Ingat bahwa dictionary, berbeda dari list dan tuple, mempunyai pasangan **kunci–nilai**. Perhatikan program berikut:

*LatReview10.py*

```
dd = {'nama':'joko', 'umur':21, 'asal':'surakarta'}
for kunci in dd:
        print(kunci,'<---->', dd[kunci])
```

□

### Konstruk `while`

Konstruk `while` dapat diilustrasikan lewat contoh berikut

**Latihan 1.11** Cetaklah bilangan 0 sampai suatu bilangan tertentu beserta kuadratnya, asalkan kuadratnya itu kurang dari 200. Program di bawah bisa menjadi penyelesaiannya (ketiklah).

*LatReview11.py*

```
bil = 0
while(bil*bil<200):
    print(bil, bil*bil)
    bil = bil + 1
```

Larikan program di atas. Hasilnya akan seperti berikut

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
11 121
12 144
13 169
14 196
```

Tepat sebelum `bil*bil >= 200`, programnya keluar dari perulangan itu. Jadi bisa kamu lihat bahwa program akan mengeksekusi blok di bawah `while` kalau kondisinya terpenuhi. Kalau sudah tidak terpenuhi, dia akan keluar dari perulangan itu. □

## 1.10 Kata-kata kunci di Python

Katakunci adalah kata yang dipakai untuk kepentingan operasional suatu bahasa, dalam hal ini Python. Kamu sudah menemui banyak katakunci ini. Misalnya `for`, `in`, `if`,...

Seperti sudah kamu duga, katakunci-katakunci ini tidak boleh (dan bahkan tidak bisa) dipakai sebagai nama variabel. Daftar katakunci ini dapat dilihat dengan mengeksekusi ini:

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

## 1.11 Soal-soal untuk Mahasiswa

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

1. Buatlah suatu fungsi `cetakSiku(x)` yang akan mencetak yang berikut:

   ```
   *
   **
   ***
   ****
   *****
   ```

   Nilai `x` menunjukkan tinggi segitiga itu (gambar di atas berarti bisa didapatkan dari menjalankan `cetakSiku(5)`) Gunakan perulangan dua kali (*double loop*)!

2. Buatlah sebuah fungsi yang menerima dua integer positif, yang akan menggambar bentuk persegi empat. Contoh pemanggilan:

   ```
   >>> gambarlahPersegiEmpat(4,5)   #tombol <enter> dipencet
   @@@@@
   @   @
   @   @
   @@@@@
   ```

3. Berikut ini adalah dua soal yang saling berkaitan

   (a) Buatlah sebuah fungsi yang menerima sebuah string dan mengembalikan sebuah list yang terdiri dari dua integer. Dua integer kembalian ini adalah: jumlah huruf di string itu dan jumlah huruf vokal (huruf vokal adalah huruf hidup) di string itu. Contoh pemanggilan:

   ```
   >>> k = jumlahHurufVokal('Surakarta')
   >>> k
   (9, 4)   # Sembilan huruf, dan empat di antaranya huruf vokal
   ```

   (b) Sama dengan soal (a) di atas, tapi sekarang yang dihitung adalah huruf konsonan. Hanya ada satu baris yang berbeda di dalam kodenya! Contoh pemanggilan:

   ```
   >>> k = jumlahHurufKonsonan('Surakarta')
   >>> k
   (9, 5)   # Sembilan huruf, dan lima di antaranya huruf konsonan
   ```

4. Buatlah sebuah fungsi yang menghitung rerata sebuah array yang berisi bilangan. Rerata mempunyai rumus

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}. \tag{1.3}$$

Namun ingatlah bahwa Python memulai index dari 0. Fungsi itu harus mempunyai bentuk `rerata(x)`, dengan `x` adalah list berisi bilangan yang ingin dihitung reratanya. Jadi, pekerjaanmu akan mempunyai bentuk:

- Buat suatu file dengan isi seperti ini

```
def rerata(b):
    #  Di sini letak
    #  programmu
    #  ...
    return hasil
```

- Larikan program itu dengan memencet F5 , lalu panggil program itu seperti ini

```
rerata([1,2,3,4,5]) # hasilnya 3
g = [3,4,5,4,3,4,5,2,2,10,11,23]
rerata(g)
```

*Extra credit:* Buat pula fungsi untuk menghitung *variance* dan *standard deviation*-nya dengan prototype, secara berurutan, `variance(x)` dan `stdev(x)`.

5. Buatlah suatu fungsi untuk menentukan apakah suatu bilangan bulat adalah bilangan prima atau bukan. Untuk mudahnya, lengkapilah program di bawah ini

```
from math import sqrt as sq
def apakahPrima(n):
    n = int(n)   # Kalau pecahan, dibuang pecahannya.
    assert n>=0  # Hanya menerima bilangan non-negatif.
    primaKecil = [2,3,5,7,11]          # Kalau angkanya kecil, akan
    bukanPrKecil = [0,1,4,6,8,9,10]# tertangkap di sini.
    if n in primaKecil:
        return True
    elif n in bukanPrKecil:
        return False
    else:
        for i in range(2,int(sq(n))+1): #Cukup sampai akar nya.
            .....          # Tugasmu
                .....      # mengisi
        .....              # titik-titik ini.
```

Setelah selesai, larikan program di atas dan lalu tes di Python Shell:

```
apakahPrima(17)
apakahPrima(97)
apakahPrima(123)
```

6. Buatlah suatu program yang mencetak semua bilangan prima dari 2 sampai 1000. Kamu *tidak* harus memanfaatkan fungsi di atas[6].

7. Buatlah suatu program yang menerima bilangan bulat positif dan memberikan *faktorisasi-prima*-nya. Faktorisasi prima adalah pemfaktoran suatu bilangan bulat ke dalam bilangan-bilangan prima yang menjadi konstituennya. Contoh:

[6] Salah satu algoritma yang bisa dipakai adalah *Sieve of Eratosthenes*. Silakan lihat di Wikipedia.

```
>>> faktorPrima(10)
(2, 5)
>>> faktorPrima(120)
(2, 2, 2, 3, 5)
>>> faktorPrima(19)
(19,)
```

8. Buat suatu fungsi `apakahTerkandung(a,b)` yang menerima dua string `a` dan `b`, lalu menentukan apakah string `a` terkandung dalam string `b`. Eksekusinya seperti ini:

```
>>> h = 'do'
>>> k = 'Indonesia tanah air beta'
>>> apakahTerkandung(h,k)
True
>>> apakahTerkandung('pusaka',k)
False
```

9. Buat program untuk mencetak angka dari 1 sampai 100. Kalau angkanya pas kelipatan 3, cetak 'Python'. Kalau pas kelipatan 5, cetak 'UMS'. Kalau pas kelipatan 3 sekaligus kelipatan 5, cetak 'Python UMS'. Jadi hasilnya:

```
1
2
Python
4
UMS
Python
7
8
Python
UMS
11
Python
13
14
Python UMS
16
17
...
```

10. Buat modifikasi pada Contoh 1.4, agar bisa menangkap kasus di mana determinannya kurang dari nol. Jika ini terjadi, tampilkan peringatan di layar seperti ini:

```
>>> selesaikanABC(1,2,3)
Determinannya negatif.  Persamaan tidak mempunyai akar real.
>>>
```

11. Buat suatu fungsi `apakahKabisat()` yang menerima suatu angka (tahun). Jika tahun itu kabisat, kembalikan `True`. Jika bukan kabisat, kembalikan `False`.

    Tahun kabisat – tahun yang memiliki tanggal 29 Februari – adalah tahun yang habis dibagi 4, kecuali dia habis dibagi 100 (maka dia bukan tahun kabisat). Tapi kalau dia habis dibagi 400, dia adalah tahun kabisat (meski habis dibagi 100).

    Berikut ini adalah beberapa contoh:

- 1896 tahun kabisat (habis dibagi 4)
- 1897 bukan tahun kabisat (sudah jelas)
- 1900 bukan tahun kabisat (meski habis dibagi 4, tapi habis dibagi 100, dan tidak habis dibagi 400)
- 2000 tahun kabisat (habis dibagi 400)
- 2004, 2008, 2012, 2016, ..., 2096 tahun kabisat
- 2100, 2200, 2300 bukan tahun kabisat
- 2400 tahun kabisat

12. Program permainan tebak angka. Buat program yang alurnya secara global seperti ini:
    - Komputer membangkitkan bilangan bulat random antara 1 sampai 100. Nilainya disimpan di suatu variabel dan tidak ditampilkan ke pengguna.
    - Pengguna diminta menebak angka itu, diinputkan lewat keyboard.
    - Jika angka yang diinputkan terlalu kecil atau terlalu besar, pengguna mendapatkan umpan balik dari komputer ("Angka itu terlalu kecil. Coba lagi")
    - Proses diulangi sampai angka itu tertebak atau sampai sekian tebakan meleset[7].

    Ketika programnya dilarikan, prosesnya kurang lebih seperti di bawah ini

    ```
    Permainan tebak angka.
    Saya menyimpan sebuah angka bulat antara 1 sampai 100. Coba tebak.
    Masukkan tebakan ke-1:> 50
    Itu terlalu kecil. Coba lagi.
    Masukkan tebakan ke-2:> 75
    Itu terlalu besar. Coba lagi.
    Masukkan tebakan ke-3:> 58
    Ya. Anda benar
    ```

13. Buat suatu fungsi `katakan()` yang menerima bilangan bulat positif dan mengembalikan suatu string yang merupakan pengucapan angka itu dalam Bahasa Indonesia. Contoh:

    ```
    >>> katakan(3125750)
    'Tiga juta seratus dua puluh lima ribu tujuh ratus lima puluh'
    ```

    Batasi inputnya agar lebih kecil dari satu milyar. *Extra credit:* gunakan rekursi.

14. Buat suatu fungsi `formatRupiah()` yang menerima suatu bilangan bulat positif dan mengembalikan suatu string yang merupakan bilangan itu tapi dengan 'format rupiah'. Contoh:

    ```
    >>> formatRupiah(1500)
    'Rp 1.500'
    >>> formatRupiah(2560000)
    'Rp 2.560.000'
    ```

[7] Kalau angka yang harus ditebak adalah antara 1 dan 100, seharusnya maksimal jumlah tebakan adalah 7. Dapatkah kamu melihat kenapa? Kalau antara 1 dan 1000, maksimal jumlah tebakan adalah 10. Apa polanya?

# Modul 2

# Mengenal OOP pada Python

Di bab ini akan kita bahas sekilas pemrograman berorientasi obyek dengan Python. Pemrograman berorientasi obyek (*object-oriented programming*, OOP) adalah sebuah konsep *powerful* yang berguna dalam pembuatan program secara modular.

Sebelum membahas lebih jauh topik OOP, kita akan membahas topik module terlebih dahulu.

## 2.1 Module

Module secara mudah bisa dipahami sebagai kumpulan prosedur dan nilai yang tersimpan dalam satu atau beberapa file, yang bisa diakses dengan meng-`import`-nya. Perhatikan contoh berikut.

**Latihan 2.1** Sebuah module sederhana. Ketik dan simpan file `.py` berikut.

*ModulePythonPertamaku.py*

```
1 def ucapkanSalam():
2     print("Assalamu 'alaikum!")
3
4 def kuadratkan(x):
5     return x*x
6
7 buah = 'Mangga'
8 daftarBaju = ['batik','loreng','resmi berdasi']
9 jumlahBaju = len(daftarBaju)
```

Kembalilah ke Python Shell (JANGAN larikan programnya). Di Python Shell, ketikkan perintah import seperti berikut

```
>>> import ModulePythonPertamaku
```

Kalau berhasil, maka fungsi dan variabel di file `ModulePythonPertamaku.py` sudah termuat ke memori. Sekarang contohlah yang berikut ini

```
>>> ModulPythonPertamaku.ucapkanSalam()  # INGAT tanda kurungnya
Assalamu 'alaikum!
>>> ModulPythonPertamaku.kuadratkan(5)
25
```

```
>>> ModulPythonPertamaku.buah
'Mangga'
>>>
```

□

Dapat dilihat bahwa sebuah module mengelompokkan fungsi dan variabel dalam satu ikatan. "Ruang-nama"-nya (*name-space*) berada di ModulPythonPertamaku. Nama modulnya dibawa ke ruang-nama lokal. Sehingga untuk mengakses metode/fungsinya kita harus mengikutsertakan nama modulnya.

Menulis "`ModulPythonPertamaku`" setiap saat tampaknya bikin capek. Bisakah kita menyingkatnya? Python bisa. Seperti ini.

```
>>> import ModulPythonPertamaku as mpp
>>> mpp.ucapkanSalam()
Assalamu 'alaikum!
>>> mpp.daftarBaju
['batik', 'loreng', 'resmi berdasi']
>>> mpp.jumlahBaju
3
```

Ruang-namanya sekarang adalah[1] `mpp`.

Bagaimana kalau dari sekian ratus metode yang ditawarkan suatu modul, kita hanya memerlukan beberapa saja? Kita bisa mengambil sesuai keperluan.

```
>>> from ModulPythonPertamaku import kuadratkan, daftarBaju
>>> kuadratkan(6)
36
>>> daftarBaju
['batik', 'loreng', 'resmi berdasi']
```

**Penting:** perhatikan bahwa fungsi dan variabel yang diimport sekarang berada di ruang-nama lokal. Yakni kita bisa *memanggilnya secara langsung.*

Kita dapat juga memberi nama lain pada metode yang diimport

```
>>> from ModulPythonPertamaku import ucapkanSalam as ucap
>>> ucap()
Assalamu 'alaikum!
```

Kita dapat meng-import seluruh metode dan variabel di suatu module. Silakan coba yang berikut

```
dir()               # Melihat isi ruangnama lokal
import math as m    # Mengimpor math sebagai m
dir()               # Lihat lagi. Pastikan ada module 'm' di sana
dir(m)              # Melihat isi module 'm'
from sys import *   # Mengimpot semua yang di 'sys' ke ruangnama lokal
dir()               # Melihat isi ruangnama lokal lagi.
```

[1] `import xxx as yy` berguna tidak terutama pada kenyamanan menyingkatnya, tapi pada konsep ruang-nama, *namespace.*

## 2.2 Class dan Object

Pada bagian ini kita akan mempelajari konsep class dan object pada Python.

Di dalam Python, sebuah *class*[2] adalah sebuah konsep atau cetak biru mengenai 'sesuatu' (umumnya kata benda). Sebuah *object*[3] adalah 'sebuah class yang mewujud'[4].

Ketika sebuah class X akan diwujudkan menjadi sebuah (atau beberapa) object di memori, berarti class X itu di-instantiasi menjadi object a,b,c, dan seterusnya. Sering pula dikatakan bahwa object a,b,c adalah *instance* dari class X. Beberapa contoh akan membantu.

- Ada class Manusia. Instance dari class Manusia misalnya: mbak Sri temanmu, mas Joko tetanggamu, si Janto sepupumu, bu Ratih gurumu. (Mereka adalah object 'yang dibangkitkan' dari class yang sama). Lebih jauh lagi, tiap instance mempunyai hal-hal yang sama. Katakanlah: setiap Manusia mempunyai nama, setiap Manusia mempunyai metode 'ucapkanSalam'.

  – Class Mahasiswa bisa dibuat dengan landasan class Manusia di atas.

  – Lebih jauh lagi, class MhsTIF bisa dibuat dengan landasan class Mahasiswa barusan.

  Dalam *object-oriented programming*, ini disebut *inheritance* atau pewarisan. Semua object yang di-instantiasi dari class Mahasiswa akan mewarisi metode dan state yag dimiliki class Manusia. Semua object yang di-instantiasi dari class MhsTIF akan mewarisi metode dan state yag dimiliki class Mahasiswa dan yang dimiliki class Manusia.

- Ada class SepedaMotor (atau 'konsep', atau 'ide tentang' SepedaMotor). Instance dari class ini misalnya: sepeda motor yang kamu miliki dan semua sepeda motor yang sekarang sedang berjalan di jalanraya. Jadi, class SepedaMotor itu di-instantiasi menjadi, salah satunya, object sepeda motor yang kamu naiki.

- Ada class Pesan. Instance dari class ini adalah pesan1 dengan isi pesan 'Aku suka kuliah ini' dan pesan2 dengan isi pesan 'Aku senang struktur data'.

Sekarang mari kita tinjau beberapa contoh

**Latihan 2.2** Sebuah kelas sederhana: `Pesan`. Ketik[5] dan simpan.

[2] Secara resmi Bahasa Indonesia punya kata 'kelas', namun untuk kejelasan paparan, kita memakai kata 'class' di sini.

[3] Bahasa Indonesia punya kata 'objek' dan 'obyek' (maknanya sama). Tapi sekali lagi untuk kejelasan paparan dan pengkodean, kita pakai kata 'object' di sini.

[4] Ini mungkin mirip istilah di matakuliah filsafat, tapi percayalah, idenya sebenarnya tidak rumit.

[5] Jika ada kata yang terasa asing seperti `self` dan `__init__`, ini akan dijelaskan kemudian

*LatOOP2.py*

```
class Pesan(object):
    """
       Sebuah class bernama Pesan.
       Untuk memahami konsep Class dan Object.
    """
    def __init__(self, sebuahString):
        self.teks = sebuahString
    def cetakIni(self):
        print(self.teks)
    def cetakPakaiHurufKapital(self):
        print(str.upper(self.teks))
    def cetakPakaiHurufKecil(self):
        print(str.lower(self.teks))
    def jumKar(self):
        return len(self.teks)
    def cetakJumlahKarakterku(self):
        print('Kalimatku mempunyai',len(self.teks),'karakter.')
    def perbarui(self,stringBaru):
        self.teks = stringBaru
```

Jalankan programnya dengan memencet F5 atau meng-`import`-nya. Maka class Pesan akan sudah berada di memori dan siap di-instantiasi. Cobalah menginstansiasi ('membuat object dari') class Pesan di atas seperti berikut[6]

```
>>> pesanA = Pesan('Aku suka kuliah ini')
>>> pesanB = Pesan('Surakarta: the Spirit of Java')
```

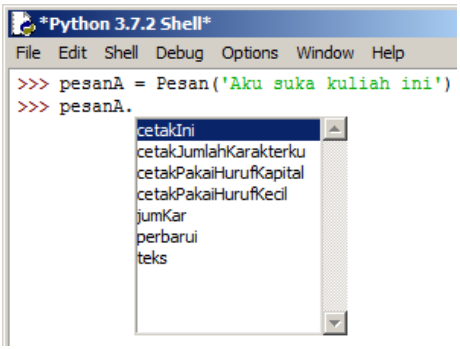
Yang barusan kamu lakukan adalah membuat dua buah object [dari class] `Pesan`, yakni `pesanA` dan `pesanB`. Object (atau variabel) ini siap dimanfaatkan sebagaimana object-object yang lain yang bertipe, misal, `int`, `str`, `float`, `bool`. Ketik yang berikut

```
>>> pesanA.cetakIni()
Aku suka kuliah ini
>>> pesanA.cetakJumlahKarakterku()
Kalimatku mempunyai 19 karakter.
>>> pesanB.cetakJumlahKarakterku()
Kalimatku mempunyai 29 karakter.
>>> pesanA.cetakPakaiHurufKapital()
AKU SUKA KULIAH INI
>>> pesanA.cetakPakaiHurufKecil()
aku suka kuliah ini
>>> pesanA.perbarui('Aku senang struktur data')
>>> pesanA.cetakIni()
Aku senang struktur data
```

□

*Cool, isn't it?* Jadi dengan class kita bisa membuat tipe data baru yang juga mempunyai metode-metode, seperti halnya tipe data bawaan. Lihat Gambar 2.1. Akan kita perjelas sedikit kejadian pembuatan class dan pemwujudan class `Pesan` di atas.

[6] Jika kamu meng-`import`-nya, boleh jadi cara memanggilnya berbeda. Tergantung caramu meng`import`.




**Gambar** 2.1: `pesanA` adalah sebuah object yang mempunyai metode-metode. Ketik `pesanA.` (jangan lupa titiknya), lalu pencet `Ctrl` + `Space` untuk melihat efeknya (di sini saya memakai IDLEX).

- Sebuah class dibuat dengan dimulai dengan kata kunci `class`, diikuti nama class-nya dengan parameter 'kelas induk'. Untuk contoh di atas kelas induknya adalah `object`.
- Sebuah class umumnya mempunyai data (seperti `teks` pada contoh di atas) dan metode (seperti `cetakJumlahKarakterku()` pada contoh di atas). Data dan metode ini bisa satu, bisa banyak, dan bahkan bisa tidak ada sama sekali.
- Penulisan metode sama dengan penulisan fungsi pada umumnya. Metode pada dasarnya adalah fungsi yang diikat pada sebuah class.
- Terdapat beberapa data dan metode khusus yang ditandai dengan awalan dan akhiran dua garis datar. Dua di antaranya:
  - `__init__()`. Ini adalah *constructor*. Ketika pembuatan object terjadi, metode inilah (kalau ada) yang dipanggil.
  - `__doc__`. Ini adalah dokumentasi. Coba ketik ini: `pesanA.__doc__`
- **Sebuah object di-instantiasi dengan mengetik nama class-nya**[7], dengan parameter sesuai dengan yang ada di metode `__init__()`.
  - `pesanA = Pesan('Aku suka kuliah ini')`
  - `pesanB = Pesan('Surakarta: the Spirit of Java')`
- Kata `self` maksudnya adalah mengacu pada diri si instance itu. Jadi kalau di class `Pesan` kita ketikkan `self.teks`, maka saat class-nya terinstantiasi menjadi object `pesanA`, kita jadi punya variable `pesanA.teks`. Cobalah!
- Sebuah metode di suatu class umumnya mempunyai bentuk

```
class sembarangKelas(object):
    def metodeSatu(self):
        pass
```

[7] To be exact, ini hanya salah satu cara. Ada cara-cara yang lain.

```
    def metodeSembilan(self,stringBaru):
        pass
```

Kata `self` ini selalu ada di situ, mengacu pada dirinya. Saat memanggil metode ini, kata `self` ini **jangan dihitung**. Jadi untuk dua di atas pemanggilannya adalah

```
>>> obQ = sembarangKelas()                    # instantiasi
>>> obQ.metodeSatu()                          # pemanggilan metodeSatu()
>>> obQ.metodeSembilan('Aku suka mie ayam') #
```

Bandingkanlah dengan class `Pesan` yang telah kamu buat.

- Kita dapat mengubah nilai suatu data dengan memanggil metode tertentu yang dibuat untuk itu. Dalam class Pesan itu, kita mempunyai metode

  ```
  def perbarui(self,stringBaru):
      self.teks = stringBaru
  ```

  yang mengubah variabel `teks` milik instance yang relevan. Misal, seperti sudah dilakukan di atas, `pesanA.perbarui('Aku senang struktur data')`.

**Latihan 2.3** Sebuah kelas sederhana lainnya. Ketik dan simpan yang berikut ini.

*LatOOP3.py*

```
1  class Manusia(object):
2      """ Class 'Manusia' dengan inisiasi 'nama' """
3      keadaan = 'lapar'
4      def __init__(self,nama):
5          self.nama = nama
6      def ucapkanSalam(self):
7          print("Salaam, namaku", self.nama)
8      def makan(self, s):
9          print("Saya baru saja makan", s)
10         self.keadaan = 'kenyang'
11     def olahraga(self,k):
12         print("Saya baru saja latihan", k)
13         self.keadaan = 'lapar'
14     def mengalikanDenganDua(self,n):
15         return n*2
16
17 ## Kali ini melarikannya lewat file yang sama.
18 ## Lewat python shell juga bisa.
19 p1 = Manusia('Fatimah')
20 p1.ucapkanSalam()
```

Larikan program di atas. Maka kamu akan mendapatkan

```
Salaam, namaku Fatimah
```

Kamu juga dapat membuat instance yang lain dan mengetes-nya. Seperti ini:

```
>>> p2 = Manusia('Budi')
>>> p2.ucapkanSalam()
Salaam, namaku Budi
```

□

Perhatikan bahwa class `Manusia` di atas mempunyai beberapa metode lain. Ada metode yang mengubah kedaan di dalam object (*internal state*), ada pula metode yang mengembalikan sesuatu. Jalankan contoh berikut di Python Shell -mu.

```
>>> ak = Manusia('Abdul Karim')
>>> ak.ucapkanSalam()
Salaam, namaku Abdul Karim
>>> ak.keadaan
'lapar'
>>> ak.makan('nasi goreng')
Saya baru saja makan nasi goreng
>>> ak.keadaan
'kenyang'
>>> ak.olahraga('renang')
Saya baru saja latihan renang
>>> ak.keadaan
'lapar'
>>> ak.makan('bakso')
Saya baru saja makan bakso
>>> ak.keadaan
'kenyang'
>>> ak.mengalikanDenganDua(8)
16
```

Bagaimana cara mengubah `keadaan` dari `'lapar'` ke `'kenyang'`? Metode apa yang membuat keadaan menjadi `'lapar'` lagi? Metode apa yang mengembalikan sesuatu?

### 2.2.1 Pewarisan

Pewarisan atau *inheritance* adalah pembuatan suatu class berdasarkan class lain. Ini adalah topik yang luas sekali, kami paparkan di sini sebagai perkenalan.

Contoh berikut memberi gambaran konsep pewarisan ini. Di sini kita akan membuat class `Mahasiswa` yang dibangun dari class `Manusia`. Ingat, seorang mahasiswa adalah juga seorang manusia.

Class Mahasiswa ini dibangun dari class manusia. Berarti dia mewarisi semua hal yang dipunyai Manusia. Class Mahasiswa bisa mempunyai metode lain yang tidak dipunyai manusia lain pada umumnya. Plus, class mahasiswa bisa saja mempunyai metode yang menutupi metode tertentu – yang sama namanya – sebagai manusia.

**Latihan 2.4** Mari kita sebuah class yang bisa menampung data-data mahasiswa: nama, NIM, kotaTinggal, uangSaku. Ketika dipanggil pertama kali untuk membuat instance seorang mahasiswa, data-data ini akan sudah disediakan.

Class ini bisa dibuat di file yang sama dengan file yang memuat class Manusia. Atau di bagian awalnya meng-import file latihan sebelumnya. Class Mahasiswa ini mewarisi semua metode dari class manusia. Juga mewarisi semua state manusia. Akan tetapi di sini ada dua metode yang menutupi metode (dengan nama yang sama) di kelas manusia. Jika metodenya dipanggil, yang dijalankan adalah metode yang paling dekat ke dirinya sendiri.

*LatOOP4.py*

```
1 class Mahasiswa(Manusia):
```

```
    """Class Mahasiswa yang dibangun dari class Manusia."""
    def __init__(self,nama,NIM,kota,us):
        """Metode inisiasi ini menutupi metode inisiasi di class Manusia."""
        self.nama = nama
        self.NIM = NIM
        self.kotaTinggal = kota
        self.uangSaku = us
    def __str__(self):
        s = self.nama + ', NIM ' + str(self.NIM) \
            + '. Tinggal di ' + self.kotaTinggal \
            + '. Uang saku Rp ' + str(self.uangSaku) \
            + ' tiap bulannya.'
        return s
    def ambilNama(self):
        return self.nama
    def ambilNIM(self):
        return self.NIM
    def ambilUangSaku(self):
        return self.uangSaku
    def makan(self,s):
        """Metode ini menutupi metode 'makan'-nya class Manusia.
        Mahasiswa kalau makan sambil belajar."""
        print("Saya baru saja makan",s,"sambil belajar.")
        self.keadaan = 'kenyang'

# ada kelanjutannya (lihat di "Soal-soal untuk Mahasiswa").
```

Untuk mengetesnya, larikan (atau `import`) script di atas lalu eksekusi yang berikut

```
m1 = Mahasiswa('Jamil',234,'Surakarta',250000)
m2 = Mahasiswa('Andi',365,'Magelang',275000)
m3 = Mahasiswa('Sri', 676,'Yogyakarta',240000)
```

Sekarang di memori sudah termuat tiga 'object mahasiswa' dengan tiap-tiap object itu mempunyai data dan metode-metode. Contoh dan kembangkan yang berikut ini:

```
>>> m1.ambilNama()
'Jamil'
>>> m2.ambilNIM()
365
>>> m3.ucapkanSalam()
Salaam, namaku Sri
>>> m3.keadaan
'lapar'
>>> m3.makan('gado-gado')
Saya baru saja makan gado-gado sambil belajar.
>>> m3.keadaan
'kenyang'
>>> print(m3)
Sri, NIM 676. Tinggal di Yogyakarta. Uang saku Rp 240000 tiap bulannya.
```

Perhatikan dua baris terakhir di atas. Bagaimana mungkin perintah `print(m3)` menghasilkan string seperti itu? Jawabannya adalah karena kita telah meng-override metode bawaan `__str__()` di class itu. Metode inilah yang dieksekusi oleh python ketika object yang bersangkutan diminta mengeluarkan suatu string[8], misal karena perintah `print()` atau 'dipaksa jadi string'

[8]Contoh di atas sebenarnya tidak begitu umum dalam penggunaan `__str__()`. Umumnya, metode ini akan

dengan *cast* `str(m3)`. Metode `__str__()` ini kegunaannya kurang lebih sama dengan metode *`toString()`* di Java. □

Dari contoh-contoh di atas, kamu sekarang sudah mengenal beberapa metode yang spesial, seperti `__init__()`, `__doc__`, `__str__()`. Masih banyak lagi metode-metode seperti ini. Silakan baca dokumentasi dan buku referensi Python yang bagus.

Contoh pewarisan lagi. Misal kita akan membuat class MhsTIF, yakni class khusus mahasiswa teknik informatika. Tapi mahasiswa teknik informatika kan juga mahasiswa? (Dan berarti juga manusia.) Kalau begitu kita pakai saja class `Mahasiswa` di atas sebagai basis.

**Latihan 2.5** Membuat class `MhsTIF` yang didasarkan pada class `Mahasiswa`.

```
1 import LatOOP4              # Atau apapun file-nya yang kamu buat tadi
2 class MhsTIF(Mahasiswa):    # perhatikan class induknya: Mahasiswa
3     """Class MhsTIF yang dibangun dari class Mahasiswa"""
4     def katakanPy(self):
5         print('Python is cool.')
```

Sekarang **semua object yang di-instantiasi dari class `MhsTIF` akan mempunyai metode dan atribut yang sama dengan metode dan atribut class induknya, `Mahasiswa`**. Tapi beda dengan mahasiswa lainnya, mahasiswa teknik informatika mempunyai metode yang hanya dimiliki mereka, yakni `katakanPy()`. Jadi kita bisa langsung mengeksekusi yang berikut ini

```
>>> m4 = MhsTIF('Badu',334,'Sragen',230000)
>>> m4.katakanPy()
Python is cool.
>>> print(m4)
Badu, NIM 334. Tinggal di Sragen. Uang saku Rp 230000 tiap bulannya.
>>> m4.keadaan
'lapar'
>>> m4.makan('pecel')
Saya baru saja makan pecel sambil belajar.
>>> m4.keadaan
'kenyang'
>>> m4.ucapkanSalam()
Salaam, namaku Badu
```

□

## 2.3 Object dan List

Seperti halnya object-object yang berasal dari class lain seperti `int`, `str`, `float`, object yang kamu buat di atas bisa juga dikumpulkan di dalam suatu list.

mengeluarkan sesuatu yang simple, yang kalau dengan contoh di atas dia akan mengeluarkan, misal, nama si mahasiswa. Selengkapnya:

```
def __str__(self):
    return self.nama
```

**Latihan 2.6** Daftar mahasiswa. Masih melanjutkan Contoh 2.4 di atas, kita sekarang akan mencoba mengumpulkan semua object mahasiswa di atas dalam suatu list. Kamu bisa membuat tambahan object mahasiswa lain terlebih dahulu. Contohlah yang berikut ini:

```
>>> daftar = [m1, m2, m3]  # tambahkan lainnya jika kamu punya
>>> for i in daftar: print(i.NIM) # tekan <Enter> dua kali

234
365
676
>>> for i in daftar: print(i)

Jamil, NIM 234. Tinggal di Surakarta. Uang saku Rp 250000 tiap bulannya.
Andi, NIM 365. Tinggal di Magelang. Uang saku Rp 275000 tiap bulannya.
Sri, NIM 676. Tinggal di Yogyakarta. Uang saku Rp 240000 tiap bulannya.
>>>
>>> daftar[2].ambilNama()
'Sri'
```

□

Sekarang, kita bisa mencoba hal-hal seperti berikut

- Dari suatu daftar mahasiswa, carilah mahasiswa yang namanya 'Sri'.
- Urutkan daftar mahasiswa itu berdasarkan NIM.

Itu adalah beberapa hal di antara yang insya Allah akan dipelajari di pertemuan-pertemuan berikutnya.

## 2.4 Class sebagai *namespace*

Sesudah sebuah class di-instantiasi menjadi suatu object, object tersebut dapat 'digantoli' dengan berbagai macam variabel.

**Latihan 2.7** Ketik dan larikan file ini

*LatOOP7.py*

```
class kelasKosongan(object):
    pass

## Sekarang kita coba
k = kelasKosongan()
k.x = 23
k.y = 47
print(k.x + k.y)
k.mystr = 'Indonesia'
print(k.mystr)
```

Bagaimanakah hasilnya? □

Seperti sudah kamu lihat di atas, variabel-variabel bisa dicantholkan dengan bebas pada instance suatu class. Feature ini merupakan salah satu kekuatan Python, meski bagi yang datang dari Java atau C++ ini juga merupakan suatu kejutan.

## 2.5 Topik berikutnya di OOP

Topik-topik yang dibahas di atas merupakan pengenalan awal OOP. Masih banyak sekali topik-topik yang harus kamu pelajari untuk menjadi mahir dalam menggunakan konsep ini. Tujuan modul ini adalah untuk memotivasti dan memberi paparan awal pada mahasiswa akan paradigma pemorgraman berorientasi objek dan mempersiapkan mahasiswa untuk menghadapi bahan-bahan praktikum selanjutnya.

Untuk belajar lebih lanjut terkait topik ini, silakan buka

- docs.python.org/3/tutorial/classes.html
- www.tutorialspoint.com/python3/python_classes_objects.htm

## 2.6 Soal-soal untuk Mahasiswa

1. Pada Contoh 2.2, kita telah membuat class Pesan yang berisi beberapa metode. Tambahkan metode-metode di bawah ini ke dalam class itu.

   (a) Metode untuk memeriksa apakah suatu string terkandung di object Pesan itu. Seperti ini hasilnya:

   ```
   >>> p9 = Pesan('Indonesia adalah negeri yang indah')
   >>> p9.apakahTerkandung('ege')
   True
   >>> p9.apakahTerkandung('eka')
   False
   ```

   (b) Metode untuk menghitung jumlah konsonan.

   ```
   >>> p10 = Pesan('Surakarta')
   >>> p10.hitungKonsonan()
   5
   ```

   (c) Metode untuk menghitung jumlah huruf vokal.

   ```
   >>> p10.hitungVokal()
   4
   ```

2. Lihat kembali contoh 2.4. Tambahkan beberapa metode seperti dijelaskan di bawah ini

   (a) Metode untuk mengambil kota tempat tinggal si mahasiswa. Seperti ini hasilnya:

   ```
   >>> m9.ambilKotaTinggal()
   'Surabaya'
   ```

   (b) Metode untuk memperbarui kota tinggal. Seperti ini hasilnya:

   ```
   >>> m9.perbaruiKotaTinggal('Sleman')
   >>> m9.ambilKotaTinggal()
   'Sleman'
   ```

   (c) Metode untuk **menambah** uang saku. Seperti ini hasilnya:

   ```
   >>> m7.ambilUangSaku()
   270000
   ```

```
>>> m7.tambahUangSaku(50000)
>>> m7.ambilUangSaku()
320000
```

3. Masih di contoh 2.4. Buatlah suatu program untuk memasukkan data mahasiswa baru lewat Python Shell secara interaktif. Seperti sudah kamu duga, gunakanlah `input()`.

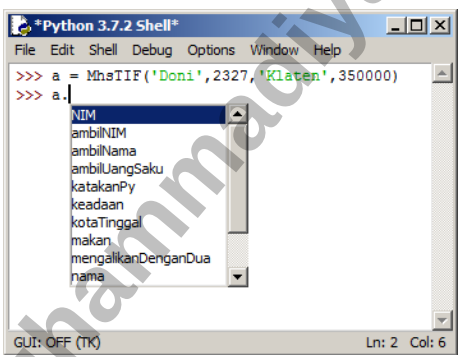
4. Buatlah state baru di class `Mahasiswa` bernama `listKuliah` yang berupa list berisi daftar matakuliah yang diambil. Buat pula metode `ambilKuliah()` yang akan menambah daftar matakuliah ini. Contoh pemanggilan:

```
>>> m234.listKuliah
[]
>>> m234.ambilKuliah('Matematika Diskrit')
>>> m234.listKuliah
['Matematika Diskrit']
>>> m234.ambilKuliah('Algoritma dan Struktur Data')
>>> m234.listKuliah
['Matematika Diskrit', 'Algoritma dan Struktur Data']
>>>
```

5. Berkaitan dengan nomer sebelumnya, buatlah metode untuk menghapus sebuah matakuliah dari `listKuliah`.

6. Dari class `Manusia`, buatlah sebuah class `SiswaSMA` yang memuat metode-metode baru (kamu bebas menentukan).

7. Dengan membuat suatu instance dari class MhsTIF (halaman 25), beri keterangan pada setiap metode dan *state* yang tampak di object itu (lihat gambar di bawah): apakah metode/*state* itu berasal dari class `Manusia`, `Mahasiswa`, atau `MhsTIF`?



# 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`.

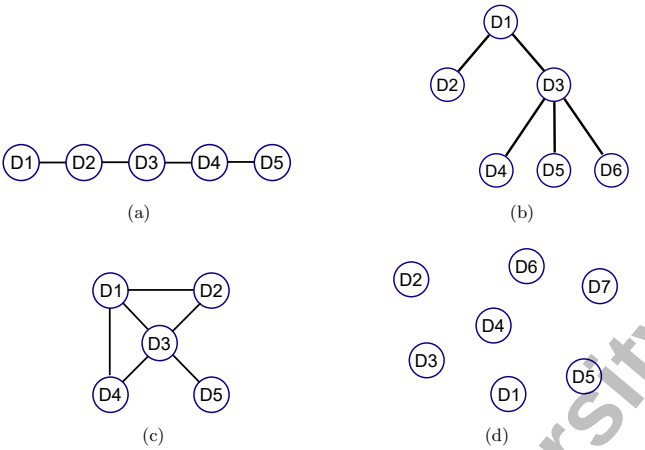
Variabel-variabel[1] ini, sebagai wadah data, mempunyai 'jenis'. Tergantung 'jenis barang' yang disimpannya. Di pelajaran-pelajaran terdahulu sudah kita lihat beberapa tipe data di Python, seperti

- `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' $2 \times 2$ 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 $3 \times 3$ 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 $3 \times 3$

```
>>> [ [ 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.

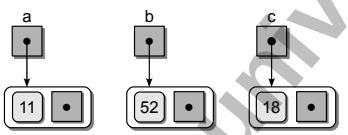
Kita mulai dengan membuat class yang akan menjadi objek-objek yang saling dihubungkan itu.

```
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)
```

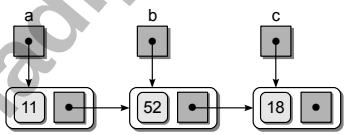
Hasilnya adalah tiga variabel dengan tiga objek[6]:



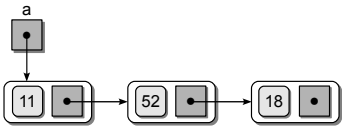

Mari kita sambungkan:

```
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:

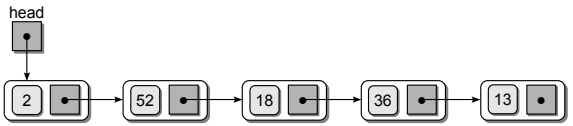



Kita bisa mengakses tiap-tiap simpul seperti ini (cobalah!):

```
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:

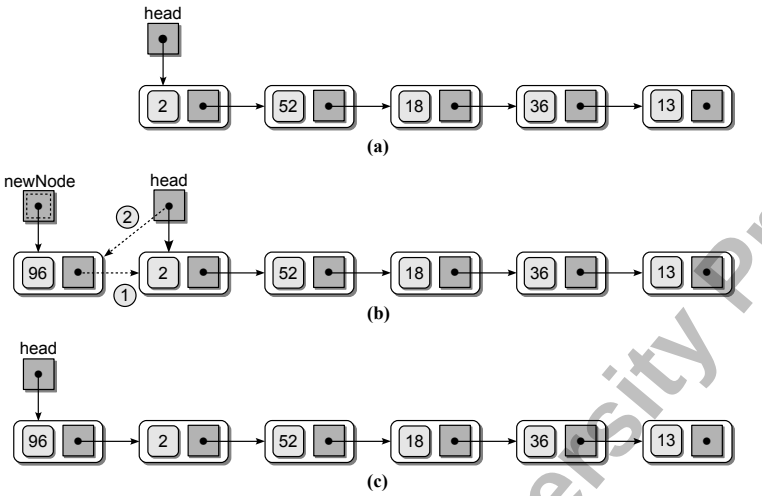
```
kunjungi(a)
```

Bagaimanakah hasilnya?

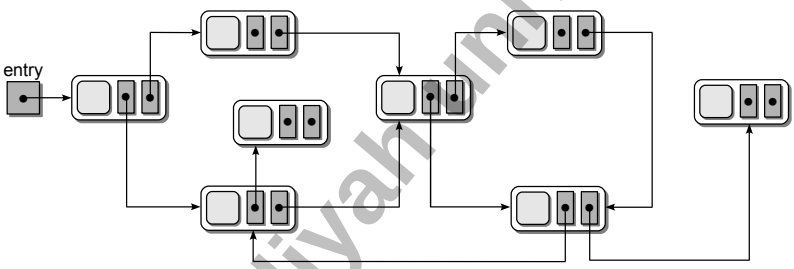
### Menambah sebuah simpul di awal

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*.

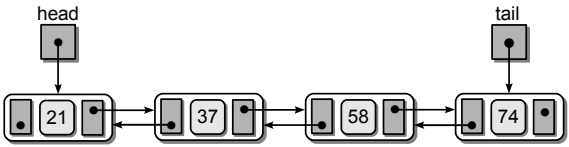
### Advanced Linked List

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

# Modul 4

# Pencarian

Algoritma pencarian, *search algorithm*, adalah salah satu algoritma yang paling sering dijalankan oleh sistem komputer maupun penggunanya. Di modul ini akan kita bahas beberapa algoritma pencarian. Di sini kita akan membahas pencarian pada struktur data satu dimensi yang *iterable*, dan tertarik pada dua kasus: yang elemenya tidak urut dan yang sudah terurutkan.

## 4.1 Linear Search

Solusi paling mudah untuk pencarian di suatu daftar adalah pencarian lurus, yakni *linear search*/*sequential search*[1]. Cara ini akan mengiterasi sepanjang daftar itu, satu item demi satu item sampai item yang dicari ditemukan atau semua item sudah diperiksa. Di Python, menemukan suatu item di sebuah list –atau apapun yang *iterable*– dapat dilakukan dengan kata kunci `in`:

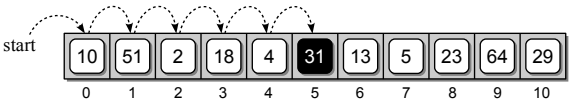
```
1 if target in arrayTempatYangDicari:
2     print("targetnya terdapat di array itu.")
3 else:
4     print("targetnya tidak terdapat di array itu.")
```

Penggunaan katakunci `in` merupakan fasilitas yang memudahkan pencarian, namun itu juga menyembunyikan kerja internalnya. Di level bawahnya, operator `in` ini diimplementasikan menggunakan `linear search`. Misal kita mempunyai array ***tidak urut*** yang didefinisikan memakai list dengan perintah berikut:

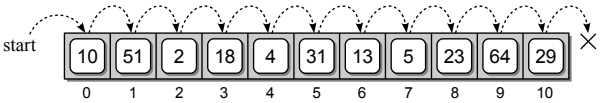
```
A = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

Untuk menentukan apakah, misal, nilai 31 terdapat pada array itu, pencarian dilakukan dengan melihat elemen pertamanya. Karena di elemen pertama tidak ketemu, pencarian dilanjutkan dengan elemen kedua. Demikian seterusnya sampai angka 31 ditemukan pada elemen keenam.

[1] Di beberapa kesempatan di modul ini, kita menerjamahkan *linear search* secara bebas dengan "pencarian lurus"




(a) Mencari 31




(b) Mencari 8

**Gambar** 4.1: Pencarian linear pada sebuah array yang tidak urut. (a) Item yang dicari ketemu. (b) Item yang dicari tidak ketemu.

Lihat Gambar 4.1(a)[2]. Bagaimana kalau item yang dicari tidak ada di array itu? Misalnya kita ingin mencari nilai 8 di array itu. Seperti sebelumnya, pemeriksaan dimulai dari elemen pertama, tapi kali ini setiap dan semua item dicocokkan dengan angka 8. Baru ketika elemen terakhir diperiksa, diketahui bahwa tidak ada angka 8 di array itu, seperti diilustrasikan di Gambar 4.1(b).

Ok? K. Nah, program berikut akan mencari `target` di data yang ada di `wadah`. Ketik dan cobalah.

```
1 def cariLurus( wadah, target ):
2     n = len( wadah )
3     for i in range( n ):
4         if wadah[i] == target:
5             return True
6     return False
```

Mari kita coba di Python Shell, seperti ini

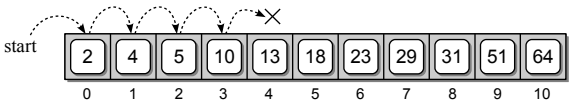
```
>>> A = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
>>> cariLurus(A,31)
True
>>> cariLurus(A,8)
False
```

Array yang urut tentu saja bisa juga dicari item-itemnya dengan linear search ini, namun dengan suatu tambahan keuntungan bahwa jika item yang diperiksa sudah 'kelewatan' dibandingkan target yang dicari, maka pencarian bisa dihentikan. Lihat Gambar 4.2.

### Pencarian Lurus untuk Objek Buatan Sendiri

Di Modul 2 kita telah belajar bagaimana membuat objek dari class yang kita buat sendiri. Kita juga dapat mencari sesuatu yang spesifik di daftar objek-objek ini. Misal kita mempunyai data mahasiswa yang diinput seperti berikut (ketiklah):

[2]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.




**Gambar** 4.2: Linear search pada data yang sudah urut. Di sini kita mencoba mencari angka 8; ketika satu persatu diperiksa dari yang paling kecil dan lalu ketemu angka 10, pencarian dihentikan.

```
c0 = MhsTIF('Ika',10,'Sukoharjo', 240000)
c1 = MhsTIF('Budi',51,'Sragen', 230000)
c2 = MhsTIF('Ahmad',2,'Surakarta', 250000)
c3 = MhsTIF('Chandra',18,'Surakarta', 235000)
c4 = MhsTIF('Eka',4,'Boyolali', 240000)
c5 = MhsTIF('Fandi',31,'Salatiga', 250000)
c6 = MhsTIF('Deni',13,'Klaten', 245000)
c7 = MhsTIF('Galuh',5,'Wonogiri', 245000)
c8 = MhsTIF('Janto',23,'Klaten', 245000)
c9 = MhsTIF('Hasan',64,'Karanganyar', 270000)
c10 = MhsTIF('Khalid',29,'Purwodadi', 265000)
##
## Lalu kita membuat daftar mahasiswa dalam bentuk list seperti ini:
##
Daftar = [c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10]
```

Kita akan mencari, sebagai contoh, mahasiswa yang beralamat Klaten. Dan, sebagai tambahan, diminta juga agar kita mencetak semua mahasiswa yang berasal dari Klaten, tidak hanya sekedar 'ketemu' dan 'tidak ketemu'. Bagaimana caranya?

Ketik dan larikan yang berikut ini

```
target = 'Klaten'
for i in Daftar:
    if i.kotaTinggal == target:
        print(i.nama + ' tinggal di ' + target)
```

### Pencarian Lurus di Linked-List

Pencarian di linked-list dilakukan dengan mengunjungi satu-persatu elemen yang ada di situ. Kamu bisa mengacu pada pencetakan item di linked list yang dipaparkan di halaman 35.

### Mencari nilai yang terkecil pada array yang tidak urut

Alih-alih mencari suatu target dengan nilai tertentu, terkadang kita diminta mencari nilai yang terkecil[3] dari suatu array. Seperti sebelumnya, kita akan melakukan pencarian lurus. Tapi kali ini kita memegang nilai terkecil yang terus diperbandingkan tiap kali kita mengunjungi elemen-elemen di array itu.

Untuk memulai loop ini, kita awalnya menganggap elemen pertama sebagai yang terkecil. Ketika mengunjungi elemen kedua, kita bandingkan nilai terkecil tadi dengan elemen ini. Jika elemen

[3] atau yang terbesar. Idenya sama saja.

ini lebih kecil, maka 'yang terkecil' tadi diubah nilainya. Ini terus dilakukan sampai elemen yang terakhir. Program berikut mengilustrasikannya.

```
def cariTerkecil(kumpulan):
    n = len(kumpulan)
    # Anggap item pertama adalah yang terkecil
    terkecil = kumpulan[0]
    # tentukan apakah item lain lebih kecil
    for i in range(1,n):
        if kumpulan[i] < terkecil:
            terkecil = kumpulan[i]

    return terkecil   #kembalikan yang terkecil
```

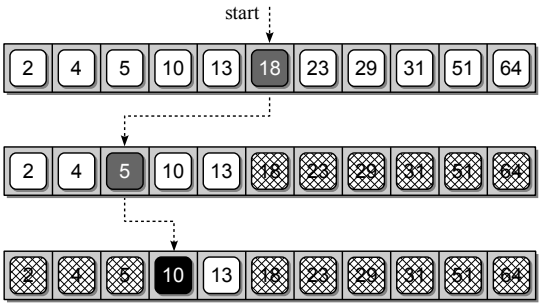
Sekarang,

- Bagaimanakah programnya jika kita ingin mencari mahasiswa (dari class MhsTIF di atas) yang uang sakunya terkecil?
- Bagaimana kalau yang terbesar?
- Bagaimanakah programnya jika kita ingin mencari *semua* mahasiswa yang uang sakunya kurang dari 250 ribu?
- Yang lebih dari 250 ribu?

## 4.2 Binary search

Jika elemen yang ada di suatu list ***sudah urut***, kita bisa memakai algoritma yang lebih efisien, yakni *binary search*. Algoritma ini bisa dijelaskan seperti di bawah ini — dengan asumsi array-nya sudah urut dari kecil ke besar.

- Dari suatu array yang urut kita ingin mencari suatu item dengan nilai tertentu.
- Ambil nilai di tengah-tengah array itu. Jika itu yang target yang dicari: selesai.
- Bandingkan nilai yang di tengah itu dengan target yang dicari.
  - Jika nilai tengah itu terlalu besar (berarti yang dicari berada di sebelah kirinya), abaikan semua yang di sebelah kanannya dan lalu cari targetnya di array yang tinggal separuh kiri itu.
  - Jika nilai tengah itu terlalu kecil (berarti yang dicari berada di sebelah kanannya), abaikan semua yang sebelah kirinya dan lalu cari targetnya di array yang tinggal separuh kanan itu.



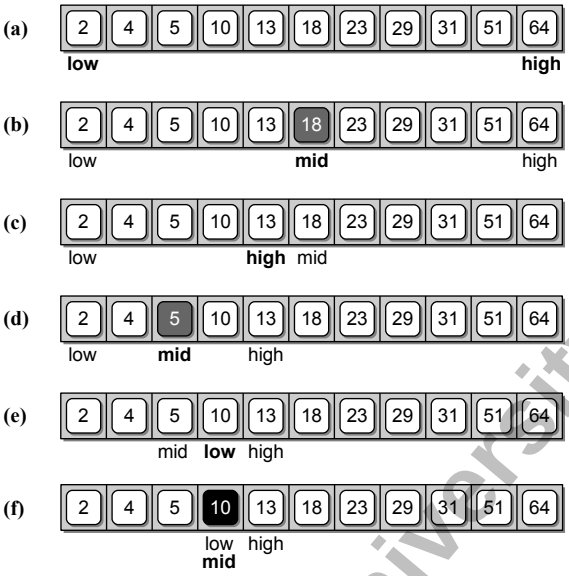

**Gambar** 4.3: Binary search. Di array yang sudah urut (dari kecil ke besar) di atas kita mencari angka 10, yakni targetnya. Pencarian dimulai dari tengah, dan bertemu angka 18. Angka ini lebih besar dari target, berarti yang dicari berada sebelah kiri, dan yang di sebelah kanan diabaikan. Sekarang ambil yang tengah lagi: ketemu angka 5. Angka ini lebih kecil dari target, jadi abaikan sebelah kirinya dan cari sebelah kanannya. Akhirnya ketemu angka 10.

Lihat Gambar 4.3 untuk ilustrasinya. Program di bawah adalah salah satu bentuk implementasi binary search.

```
def binSe(kumpulan, target):
    # Mulai dari seluruh runtutan elemen
    low = 0
    high = len(kumpulan) - 1

    # Secara berulang belah runtutan itu menjadi separuhnya
    #   sampai targetnya ditemukan
    while low <= high:
        # Temukan pertengahan runtut itu
        mid = (high + low) // 2
        # Apakah pertengahannya memuat target?
        if kumpulan[mid] == target:
            return True
        # ataukah targetnya di sebelah kirinya?
        elif target < kumpulan[mid]:
            high = mid - 1
        # ataukah targetnya di sebelah kanannya?
        else:
            low = mid + 1
    # Jika runtutnya tidak bisa dibelah lagi, berarti targetnya tidak ada
    return False
```

Program di atas akan mengembalikan `True` jika targetnya ditemukan dan `False` jika targetnya tidak ditemukan. Gambar 4.4 mengilustrasikan kejadiannya ketika kita mencari angka 10 di array di Gambar 4.3.

*Dapatkah kamu mengubah programnya agar dia mengembalikan index-nya kalau targetnya ditemukan, dan mengembalikan `False` kalau targetnya tidak ditemukan?*




**Gambar** 4.4: Langkah-langkah yang dilakukan oleh algoritma *binary search* dalam mencari angka 10: (a) rentang awal elemen-elemennya, (b) menentukan lokasi pertengahannya, (c) mengeliminasi separuh atas, (d) pertengahannya separuh bawah, (e) mengeliminasi perempat bawah, (f) menemukan target.

## 4.3 Soal-soal untuk Mahasiswa

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

1. Buatlah suatu fungsi pencarian yang, alih-alih mengembalikan `True/False`, mengembalikan *semua* index lokasi elemen yang dicari. Jadi, misal pada list daftar mahasiswa di halaman 40 kita mencari mahasiswa yang berasal dari Klaten, kita akan mendapatkan `[6, 8]`. Kalau yang dicari tidak ditemukan, fungsi ini akan mengembalikan list kosong.

2. Dari list daftar mahasiswa di atas, buatlah fungsi untuk menemukan uang saku yang terkecil di antara mereka.

3. Ubah program di atas agar mengembalikan objek mahasiswa yang mempunyai uang saku terkecil. Jika ada lebih dari satu mahasiswa yang uang sakunya terkecil, semua objek mahasiswa itu dikembalikan.

4. Buatlah suatu fungsi yang mengembalikan semua objek mahasiswa yang uang sakunya kurang dari 250000.

5. Buatlah suatu program untuk mencari suatu item di sebuah linked list.

6. Binary search. Ubahlah fungsi `binSe` di halaman 43 agar mengembalikan index lokasi elemen yang ditemukan. Kalau tidak ketemu, akan mengembalikan `False`.

7. Binary search. Ubahlah fungsi `binSe` itu agar mengembalikan semua index lokasi elemen yang ditemukan. Contoh: mencari angka 6 pada list `[2, 3, 5, 6, 6, 6, 8, 9, 9, 10,`

`11, 12, 13, 13, 14]` akan mengembalikan `[3, 4, 5]`. Karena sudah urut, "tinggal melihat kiri dan kanannya".

8. Pada permainan tebak angka yang sudah kamu buat di Modul 1 (soal nomer 12, halaman 16), kalau angka yang harus ditebak berada di antara 1 dan 100, seharusnya maksimal jumlah tebakan adalah 7. Kalau antara 1 dan 1000, maksimal jumlah tebakan adalah 10. Mengapa seperti itu? Bagaimanakah polanya?

# Modul 5

# Pengurutan

Pengurutan (*sorting*) adalah proses menyusun atau menata koleksi item-item sedemikian rupa sehingga setiap item dengan item yang sesudahnya memenuhi persyaratan hubungan tertentu. Item-item ini bisa berupa nilai-nilai yang sederhana seperti bilangan bulat dan bilangan real, atau bisa juga berupa sesuatu yang lebih kompleks seperti data rekam mahasiswa, atau entri kamus. Yang manapun saja, pengurutan item-item itu didasarkan pada nilai ***kunci pengurutan*** (*sort key*). Kunci pengurutan ini bisa berupa nilai dirinya sendiri untuk tipe data primitif (int, float, str) atau bisa juga berupa komponen tertentu atau kombinasi komponen tertentu untuk pengurutan yang lebih kompleks.

Problem dasar pengurutan dapat diilustrasikan sebagai berikut. Diberikan sebuah array:

```
[10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

buatlah suatu fungsi untuk mengurutkan elemen-elemen di array itu dari yang paling kecil ke yang paling besar, menjadi:

```
[2, 4, 5, 10, 13, 18, 23, 29, 31, 51, 64]
```

Pengurutan merupakan salah satu algoritma yang paling sering dijalankan dan sampai sekarang masih merupakan bidang ilmu yang aktif diselidiki. Gambar 5.1 mendaftari berbagai macam algoritma pengurutan yang artikelnya terdapat di Wikipedia Bahasa Inggris.

Di modul ini kita akan mempelajari beberapa algoritma pengurutan sederhana: *bubble sort*, *selection sort*, dan *insertion sort*. Insya Allah di modul selanjutnya kita akan membahas sejumlah algoritma pengurutan yang lebih kompleks (yang juga lebih cepat).

Namun sebelum membahas algoritma pengurutan itu semua, kita tulis dulu sebuah *routine* yang akan sering dipakai, yakni fungsi tukar posisi, atau *swap*, untuk menukar posisi dua elemen di suatu list.

*routine swap untuk menukar A[p] dan A[q]*

```
1 def swap(A,p,q):
2     tmp  = A[p]
3     A[p] = A[q]
4     A[q] = tmp
```

From Wikipedia, the free encyclopedia

| Sorting algorithms | |
|---|---|
| **Theory** | Computational complexity theory · Big O notation · Total order · Lists · Inplacement · Stability · Comparison sort · Adaptive sort · Sorting network · Integer sorting |
| **Exchange sorts** | Bubble sort · Cocktail sort · Odd–even sort · Comb sort · Gnome sort · Quicksort · Stooge sort · Bogosort |
| **Selection sorts** | Selection sort · Heapsort · Smoothsort · Cartesian tree sort · Tournament sort · Cycle sort |
| **Insertion sorts** | Insertion sort · Shellsort · Splaysort · Tree sort · Library sort · Patience sorting |
| **Merge sorts** | Merge sort · Cascade merge sort · Oscillating merge sort · Polyphase merge sort · Strand sort |
| **Distribution sorts** | American flag sort · Bead sort · Bucket sort · Burstsort · Counting sort · Pigeonhole sort · Proxmap sort · Radix sort · Flashsort |
| **Concurrent sorts** | Bitonic sorter · Batcher odd–even mergesort · Pairwise sorting network |
| **Hybrid sorts** | Block sort · Timsort · Introsort · Spreadsort · UnShuffle sort · JSort |
| **Other** | Topological sorting · Pancake sorting · Spaghetti sort |

**Gambar** 5.1: Sejumlah algoritma pengurutan yang artikelnya terdapat di Wikipedia bahasa inggris. Kunjungi `en.wikipedia.org/wiki/Sorting_algorithm` untuk memulainya. Di modul ini kita akan mulai dengan tiga algoritma: *bubble sort*, *selection sort*, dan *insertion sort*.

Perhatikan bahwa tidak ada kata `return` di sana. (Mengapa?) Mari kita uji routine di atas dengan program kecil seperti berikut.

```
>>> K = [50, 20, 70, 10]
>>> swap(K, 1, 3)
>>> K
[50, 10, 70, 20]
```

Berhasil! Elemen dengan index 1 dan 3 telah bertukar posisi. Kita juga akan memerlukan *routine* untuk mencari index yang nilainya terkecil, saat diberikan jangkauan indexnya[1].

*routine untuk mencari index dari elemen yang terkecil*

```
1 def cariPosisiYangTerkecil(A, dariSini, sampaiSini):
2     posisiYangTerkecil = dariSini          #-> anggap ini yang terkecil
3     for i in range(dariSini+1, sampaiSini): #-> cari di sisa list
4         if A[i] < A[posisiYangTerkecil]:    #->  kalau menemukan yang lebih kecil,
5             posisiYangTerkecil = i          #->  anggapan dirubah
6     return posisiYangTerkecil
```

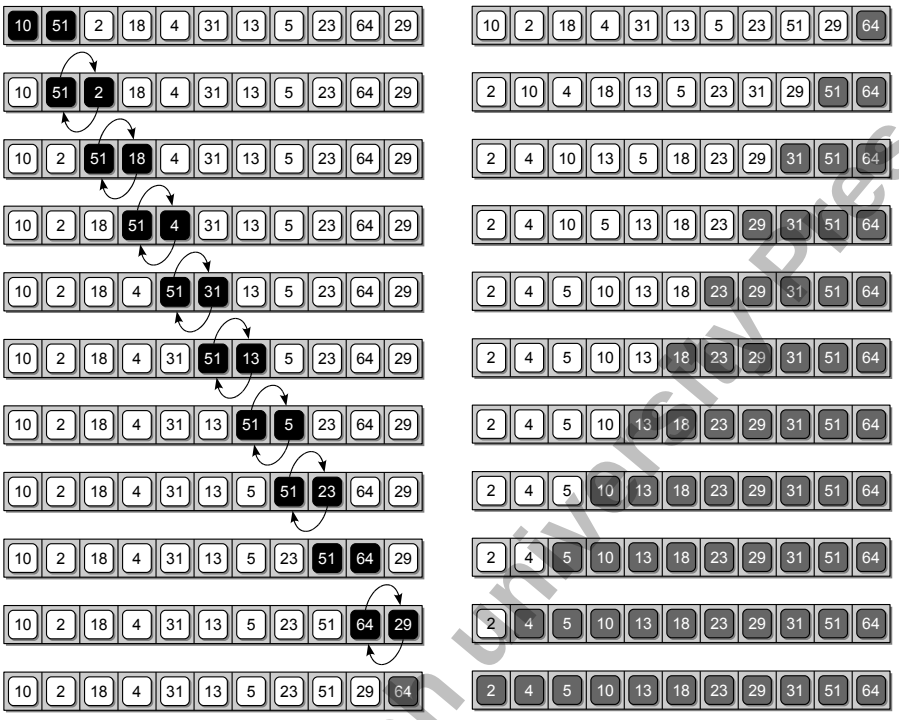
Untuk mengujinya, kita ketik yang berikut di Python Shell

```
>>> A = [18, 13, 44, 25, 66, 107, 78, 89]
>>> j = cariPosisiYangTerkecil(A, 2, len(A))
>>> j
3
```

Pada kode di atas, kita mencari index yang nilainya terkecil dalam jangkauan dari 2 sampai akhir list, hasilnya adalah index 3 (dengan nilai A[3]=25). Kalau jangkauannya dari awal sampai akhir list, hasilnya adalah index 1 (dengan nilai A[1]=13). Sekarang kita siap dengan semua algoritma pengurutan[2].

[1] Lihat juga algoritma pencarian nilai terkecil di halaman 42. Di sana kita mencari *nilai* yang terkecil. Di sini kita mencari *index* yang nilainya terkecil.

[2] Di sini kita tidak akan menggunakan fungsi dan metode bawaan Python seperti `A.sort()` atau `min(A)`. Juga, di sini kita akan tetap memakai array yang sama antara array asal dengan hasilnya. Maksudnya, array asal itulah satu-satunya array yang diproses. *Kita tidak membuat array baru.* Dalam beberapa aplikasi hal ini benar-benar




**Gambar** 5.2: Contoh penerapan ***bubble sort***. (a) ***putaran-sisi-dalam pertama*** yang lengkap, menempatkan angka 64 pada posisi yang tepat. Kotak-kotak hitam mewakili nilai yang diperbandingkan. Anak panah menunjukkan pertukaran posisi. (b) Hasil penerapan bubble sort pada runtut contoh. Setiap satu baris pada gambar (b) menunjukkan satu putaran-sisi-dalam.

## 5.1 Bubble Sort

Pengurutan dengan bubble sort merupakan salah satu algoritma pengurutan yang paling sederhana. Algoritmanya bisa diterangkan seperti berikut.

1. Diberikan sebuah list yang kita diminta mengurutkan dari kecil ke besar:

   ```
   L = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
   ```

   - Bandingkan nilai elemen pertama dengan elemen kedua. Kalau elemen pertama lebih besar dari elemen kedua, tukar posisinya.
   - Sesudah itu bandingkan elemen kedua dengan elemen ketiga. Kalau elemen kedua lebih besar dari elemen ketiga, tukar posisinya.
   - Lakukan hal seperti di atas sampai elemen terakhir. Perulangan ini kita namakan

diperlukan.

***putaran-dalam***. Perhatikan bahwa dengan cara ini elemen yang paling besar akan 'menggelembung[3] ke kanan' dan akhirnya berakhir di paling kanan.

2. Ulangi lagi hal di atas dari awal sampai elemen yang terakhir kurang satu. (Karena elemen terbesarnya sudah berada di elemen terakhir.) Pada putaran ketiga: ulangi lagi hal di atas dari awal sampai elemen yang terakhir kurang dua. Demikian seterusnya.

Gambar 5.2(a) menunjukkan putaran-dalam pertama yang lengkap sebuah bubble sort. Sedangkan Gambar 5.2(b) menunjukkan hasil masing-masing putaran dalam dan akhirnya hasil akhir penerapan bubble sort pada runtut contoh. Berikut ini adalah program Python-nya.

```
1 def bubbleSort(A):
2     n = len(A)
3     for i in range(n-1):       #-> Lakukan operasi gelembung sebanyak n-1
4         for j in range(n-i-1): #->   Dorong elemen terbesar ke ujung kanan
5             if A[j] > A[j+1]:  #->     Jika di kiri lebih besar dari di kanannya,
6                 swap(A,j,j+1)  #->     tukar posisi elemen ke j dengan ke j+1
```

Pertanyaan: dengan elemen sebanyak $n$, berapa banyakkah operasi pembandingan dan pertukaran yang dilakukan oleh algoritma bubble sort ini? Selidiki nilainya untuk *worst-case*, *average-case*, dan *best-case scenario*[4].

Algoritma bubble sort ini sangat lambat dan merupakan contoh suatu *bad algorithm*. Bahkan ada beberapa pakar[5] yang menyarankan bahwa algoritma bubble sort ini *tidak usah diajarkan* di kuliah.

## 5.2 Selection Sort

Metode pengurutan lain yang cukup sederhana adalah metode *selection sort*. Berikut ini adalah penjelasannya

1. Mulai dari $i = 0$.
2. Pegang elemen ke $i$. Dari elemen $i$ itu hingga terakhir, cari yang terkecil.
3. Kalau elemen ke $i$ itu ternyata yang terkecil, berarti dia pada posisi yang tepat dan lanjutkan ke langkah berikutnya.

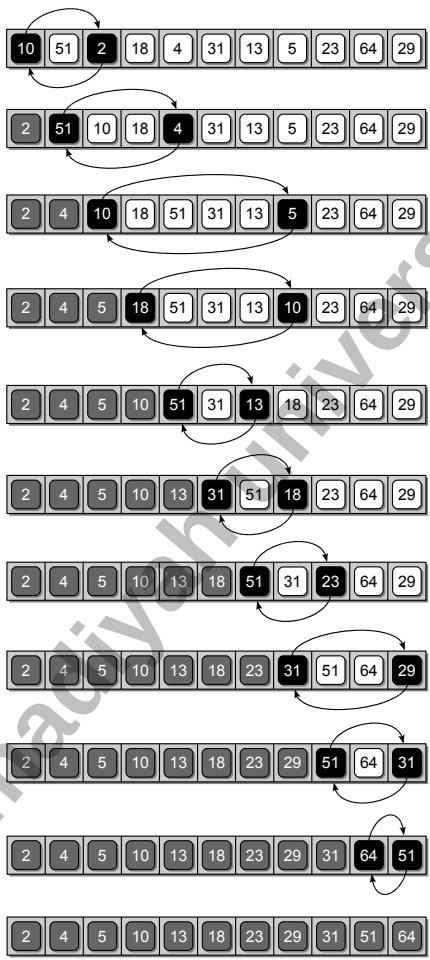
   Tapi kalau elemen terkecil itu bukan elemen ke $i$ (berarti ada di sisi yang lebih kanan), tukar posisi elemen ke $i$ dengan elemen terkecil itu tadi.
4. Lanjutkan untuk $i + 1$ dan ulangi proses 2 dan 3. Demikian seterusnya sampai elemen terakhir.

Gambar 5.3 menunjukkan hasil penerapan selection sort ini pada array contoh. Perhatikan bahwa algoritma ini lebih cepat daripada algoritma bubble sort yang kita bahas sebelumnya.

[3] Bahasa Inggris: *bubble* artinya gelembung (seperti gelembung sabun)

[4] *worst-case*: kejadian terburuk, misalnya array itu awalnya dalam keadaan urut terbalik; *average-case*: kejadian 'yang biasanya'; *best-case*: kejadian terbaik, misalnya array itu awalnya dalam keadaan sudah urut.

[5] misal Prof. Owen Astrachan dari *Duke University*




**Gambar** 5.3: Hasil penerapan algoritma pengurutan ***selection sort*** pada array contoh. Kotak-kotak berwarna abu-abu menunjukkan nilai-nilai yang sudah diurutkan. Kotak-kotak berwarna hitam menunjukkan nilai-nilai yang ditukar posisinya pada tiap iterasi di algoritma ini.

Kode berikut adalah salah satu implementasi selection sort dengan bahasa Python.

```
1 def selectionSort(A):
2     n = len(A)
3     for i in range(n - 1):
4         indexKecil = cariPosisiYangTerkecil(A, i, n)
5         if indexKecil != i :
6             swap(A, i, indexKecil)
```

Pada implementasi lain algoritma ini, pemeriksaan di baris 5 ditiadakan. Jadi setelah ditemukan posisi yang nilainya terkecil, langsung ditukar tanpa diperiksa terlebih dahulu. Cobalah untuk menyelidiki mana yang lebih cepat di antara dua implementasi ini.

Pertanyaan: dengan elemen sebanyak $n$, berapa banyakkah operasi pembandingan dan pertukaran yang dilakukan oleh algoritma selection sort ini? Selidiki nilainya untuk *worst-case*, *average-case*, dan *best-case scenario*

## 5.3 Insertion Sort

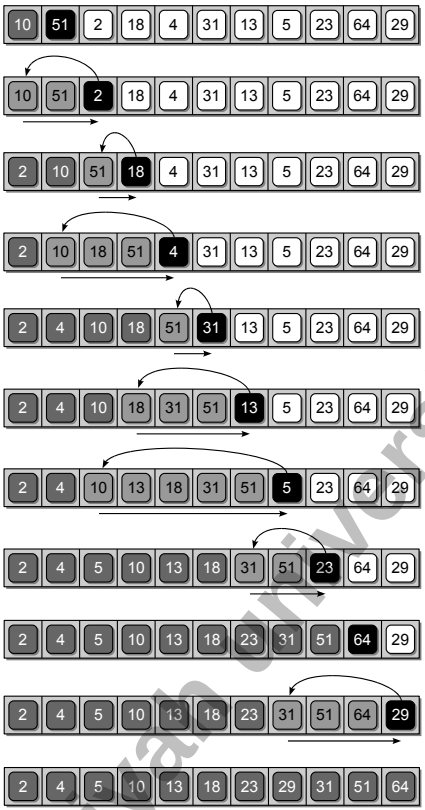
Algoritma pengurutan sederhana lain adalah algoritm *insertion sort*[6]. Algoritma ini mempunyai ide dasar seperti berikut (lihat pula Gambar 5.4):

1. Mulai dari elemen pertama. Anggap ini sebagai elemen dengan nilai terkecil.
2. Sekarang lihat elemen berikutnya (di sebelah kanannya). Tempatkan/sisipkan elemen ini ke kiri sesuai dengan nilainya.
3. Ulangi kembali proses nomer dua di atas untuk elemen di sebelah kanannya lagi. Demikian seterusnya sampai elemen terakhir diposisikan ke tempat yang tepat.

Sebagai contoh, perhatikan Gambar 5.4 di halaman 53.

- Anggap angka 10 sebagai nilai terkecil. Lalu, lihat baris pertama, angka 51 coba diposisikan ke tempat yang tepat. Tidak ada pergeseran karena 51 lebih besar dari 10.
- Baris kedua. Angka 2 coba diposisikan ke tempat yang tepat. Dia harus menggeser 10 dan 51 ke kanan agar angka 2 ini menempati posisi yang tepat.
- Baris ketiga. Angka 18 datang. Dia harus disisipkan ke tempat yang tepat. Untuk itu dia harus menggeser 51 ke kanan.
- Baris keempat. Angka 4 datang. Dia harus menggeser 10, 18, 51 ke kanan sampai dia menempati posisi yang tepat.
- ... demikian seterusnya.

[6] *Insertion* dapat diartikan *penyisipan*.




**Gambar** 5.4: Hasil penerapan algoritma pengurutan ***insertion sort*** pada array contoh. Kotak-kotak berwarna abu-abu menunjukkan nilai-nilai yang sudah diurutkan. Kotak-kotak berwarna hitam menunjukkan nilai berikutnya yang akan diposisikan. Kotak-kotak abu-abu yang lebih terang dengan tulisan hitam adalah nilai yang sudah urut yang harus digeser ke kanan untuk memberi tempat pada nilai yang sedang disisipkan.

Program di bawah ini adalah salah satu bentuk implementasi insertion sort.

```
1 def insertionSort(A):
2     n = len(A)
3     for i in range(1, n):
4         nilai = A[i]
5         pos = i
6         while pos > 0 and nilai < A[pos - 1]: # -> Cari posisi yang tepat
7             A[pos] = A[pos - 1]               #    dan geser ke kanan terus
8             pos = pos - 1                     #    nilai-nilai yang lebih besar
9         A[pos] = nilai # -> Pada posisi ini tempatkan nilai elemen ke i.
```

## 5.4 Soal-soal untuk Mahasiswa

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

1. Buatlah suatu program untuk mengurutkan array mahasiswa berdasarkan NIM, yang elemennya terbuat dari class `MhsTIF`, yang telah kamu buat sebelumnya.

2. Misal terdapat dua buah array *yang sudah urut* $A$ dan $B$. Buatlah suatu program untuk menggabungkan, secara efisien, kedua array itu menjadi suatu array $C$ yang urut.

3. Kamu mungkin sudah menduga, *bubble sort* lebih lambat dari *selection sort* dan juga *insertion sort*. Tapi manakah yang lebih cepat antara *selection sort* dan *insertion sort*?[7] Untuk memulai menyelidikinya, kamu bisa membandingkan waktu yang diperlukan untuk mengurutkan sebuah array yang besar, misal sepanjang 6000 (enam ribu) elemen.

```
from time import time as detak
from random import shuffle as kocok
k = range(1,6001)
kocok(k)
u_bub = k[:] ## \\
u_sel = k[:] ##   -- Jangan lupa simbol [:]-nya!.
u_ins = k[:] ## //

aw=detak();bubbleSort(u_bub);ak=detak();print('bubble: %g detik' %(ak-aw) );
aw=detak();selectionSort(u_sel);ak=detak();print('selection: %g detik' %(ak-aw) );
aw=detak();insertionSort(u_ins);ak=detak();print('insertion: %g detik' % (ak-aw) );
```

   Bandingkan hasil percobaan kamu dengan hasil teman-temanmu. Jika waktu untuk pengurutan dirasa terlalu cepat, kamu bisa memperbesar ukuran array itu.

   Perlu diingat bahwa hasil yang kamu dapat belum dapat menyimpulkan apa-apa. Selain strategi implementasinya, banyak kondisi yang harus diperhatikan dalam penerapan suatu algoritma. Untuk keadaan tertentu, dipakai selection sort. Untuk keadaan tertentu yang lain, dipakai insertion sort. Untuk keadaan lain lagi, dipakai algoritma yang berbeda[8].

Ketiga algoritma pengurutan yang dibahas di modul ini termasuk algoritma awal yang relatif sederhana. Para ahli telah menyelidiki dan membuat berbagai algoritma pengurutan yang lebih cepat, yang insya Allah beberapa di antaranya akan kita bahas di modul berikutnya.



[7] Setidaknya untuk versi implementasi di modul ini. Versi yang lebih optimal bisa jadi membuahkan hasil yang berbeda.

[8] Python sendiri, ketika kita memanggil `A.sort()`, memakai algoritma yang dinamai *Timsort*, yang merupakan gabungan beberapa algoritma. Lihat `http://en.wikipedia.org/wiki/Timsort` . Cobalah membandingkan perintah pengurutan di Python (misal: `k.sort()`) dengan algortima yang telah kamu buat di atas.

# Modul 6

# Pengurutan lanjutan

Di modul sebelumnya kita sudah mengenal beberapa algoritma pengurutan seperti selection sort dan insertion sort. Algoritma-algoritma itu cukup sederhana dan intuitif, namun masih dirasa belum cukup cepat oleh para ahli. Riset di bidang ini terus dilakukan. Algoritma pengurutan yang menjadi bawaan suatu bahasa pemrograman pun berubah-ubah.

Di modul ini akan kita pelajari pengurutan yang sedikit lebih *advanced*, yakni *merge sort* dan *quick sort*. Sebelumnya akan kita tinjau sebuah problem yang menarik, yakni menggabungkan dua list yang sudah urut.

## 6.1 Menggabungkan dua list yang sudah urut

Terkadang kita perlu menggabungkan dua buah list yang sudah urut menjadi sebuah list baru. Perhatikan kode-kodel berikut

```
P = [2, 8, 15, 23, 37]
Q = [4, 6, 15, 20]
R = gabungkanDuaListUrut(P, Q)
print(R)
```

yang membuat dua list –yang sudah urut kecil ke besar– lalu memanggil sebuah fungsi buatan sendiri. Fungsi ini mengembalikan sebuah list baru yang dibuat dengan menggabungkan dua list tadi. Ketika dicetak akan muncul

```
[2, 4, 6, 8, 15, 15, 20, 23, 37]
```

Bagaimanakah algoritmanya? Bisa saja kita langsung menggabungkan keduanya dan lalu memanggil sebuah algoritma yang sudah kita buat sebelumnya[1]. Namun ini berarti kita tidak memanfaatkan kenyataan bahwa kedua list ini sudah urut. Kita bisa membuat algoritma yang lebih baik.

[1] Misal seperti ini:

```
def gabungkanDuaListUrut(A, B):
    C = A + B; insertionSort(C); return C
```

Misalkan kita akan mempunyai dua tumpukan lembar jawaban ujian yang masing-masingnya sudah urut NIM. Bagaimanakah menggabungkannya? Pertama lihat kertas yang paling atas di dua tumpukan itu. Bandingkan mana yang paling kecil NIM-nya. Ambil kertas yang paling kecil NIM-nya itu lalu taruh di meja secara terbalik untuk mulai membuat tumpukan baru. Lalu kembali lihat kedua tumpukan tadi. Bandingkan kedua NIM di kertas itu. ...

Demikian seterusnya. Akan ada suatu saat di mana salah satu tumpukan sudah habis sedangkan tumpukan yang satunya masih ada. Dalam hal ini, kita tinggal menumpukkan sisa tadi ke tumpukan yang baru. Berikut ini adalah salah satu implementasinya di Python.

```
def gabungkanDuaListUrut(A, B):
    la = len(A); lb = len(B)
    C = list()      # C adalah list baru
    i = 0; j = 0

    # Gabungkan keduanya sampai salah satu kosong
    while i < la and j < lb:
        if A[i] < B[j]:
            C.append(A[i])
            i += 1
        else:
            C.append(B[j])
            j += 1

    while i < la:        # Jika A mempunyai sisa
        C.append(A[i])   #   tumpukkan ke list baru itu
        i += 1           #   satu demi satu

    while j < lb:        # Jika B mempunyai sisa
        C.append(B[j])   #   tumpukkan ke list baru itu
        j += 1           #   satu demi satu

    return C
```

Cobalah melarikan program di atas untuk semua kemungkinan penggunaan. Perlu dicatat bahwa program di atas bukan satu-satunya cara untuk menyelesaikan problemnya. Banyak cara yang lain, namun yang ini dipilih sebagai *prelude* untuk algoritma *merge sort* di bawah.

## 6.2 Merge sort

Algoritma merge sort memakai strategi *divide and conquer*, bagi dan taklukkan. Algoritma ini adalah algoritma *recursive* yang secara terus menerus membelah (men-*split*) sebuah list menjadi dua. (Kotak di halaman 57 memberikan penyegaran tentang fungsi rekursif.) Jika list-nya kosong atau hanya berisi satu elemen, maka *by definition* dia sudah urut (ini adalah *base case*-nya).

Jika list-nya mempunyai item lebih dari satu, kita membelah list-nya dan secara rekursif memanggil `mergeSort` di keduanya. Kalau kedua paruh –kanan dan kiri– sudah urut, operasi penggabungan (*merge*) dilakukan. Proses penggabungan adalah proses mengambil dua list urut yang lebih kecil dan menggabungkan mereka menjadi satu list baru yang urut. Lihat

## Penyegaran: Fungsi Rekursif

Seperti sudah kamu pelajari sebelumnya, fungsi rekursif[a] adalah fungsi yang memanggil dirinya sendiri. Sebagai contoh, misal kita diminta menghitung 7! (tujuh faktorial). Dari definisi, kita akan menghitungnya sebagai

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \tag{6.1}$$

Tapi perhatikan bahwa ini bisa juga ditulis sebagai

$$7! = 7 \times 6! \tag{6.2}$$

Tapi 6! juga bisa ditulis sebagai

$$6! = 6 \times 5! \tag{6.3}$$

Dan demikian seterusnya. Kapankah ini berakhir? Yakni saat mencapai angka 1, di mana,

$$1! = 1 \tag{6.4}$$

Sehingga ini merupakan *base case* (kejadian dasar) untuk fungsi faktorial kita. Dari sini kita tahu bahwa fungsi rekursif mempunyai tiga unsur:

- kejadian dasar (*base case*)
- kejadian rekursif (*recursive case*)
- hal yang memastikan bahwa semua *recursive cases* akan ter-reduksi ke *base case*[b].

Dari sini kita bisa membuat program faktorial di atas sebagai berikut.

```
def faktorial(a):
    if a==1:          ## base case
        return 1
    else:             ## recursive case
        return a*faktorial(a-1)
```

Perhatikan bahwa kita mempunyai kejadian dasar, kejadian rekursif, dan hal yang memastikan bahwa program ini akan konvergen. Fungsi rekursif adalah *fungsi yang memanggil dirinya sendiri*[c]. Cobalah jalankan program di atas dan periksa hasilnya.

[a] Bahasa Inggris: *recursive function*.

[b] Kalau tidak, program akan divergen, "meledak", dan tidak berakhir. Misal kalau tanda minus pada program faktorial di atas diganti dengan tanda plus.

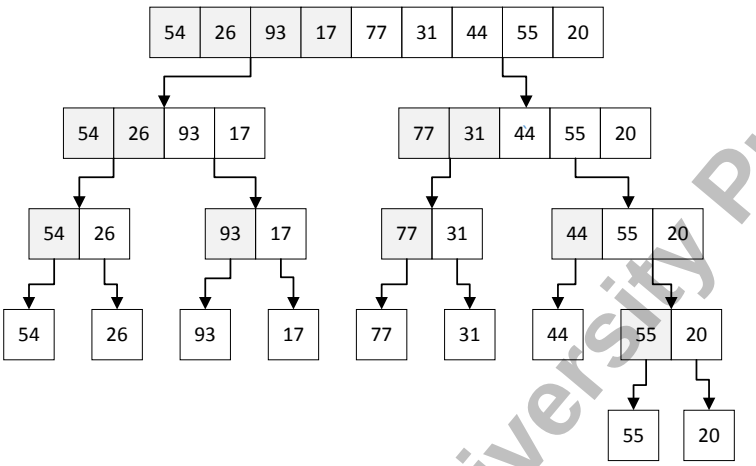
[c] Untuk paham fungsi rekursif, terlebih dahulu kamu harus paham fungsi rekursif. :-)

kembali halaman-halaman seblumnya.

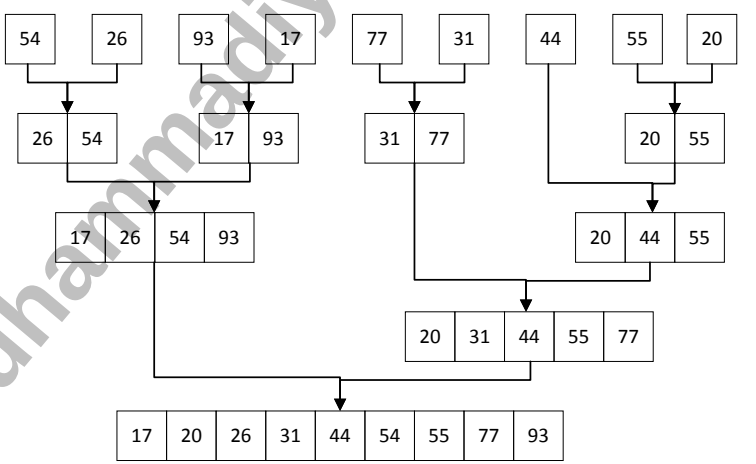
Gambar 6.1 menunjukkan[2] sebuah list yang dibelah-belah. Gambar 6.2 menunjukkan penggabungan list itu sampai urut. Perhatikan bahwa jumlah elemennya tidak harus pangkat dua (2, 4, 8, 16, ...) dan dengan demikian jumlah elemen kiri dan kanan pada suatu saat bisa berbeda. Ini tidak menjadi masalah.

Program merge sort dapat dilihat di bawah. Program ini mulai dengan menanyakan pertanyaan *base case*. Jika panjangnya list kurang dari atau sama dengan satu, kita berarti sudah mempunyai sebuah list yang urut dan tidak perlu pemrosesan lagi. Tapi, sebaliknya, jika panjang list itu lebih dari satu, kita lalu menggunakan operasi *slice* di Python untuk meng-ekstrak separuh kiri dan kanan. Perlu dicatat bahwa list-nya bisa jadi tidak mempunyai jumlah elemen yang genap. Ini tidaklah mengapa karena perbedaan jumlah elemen antara separuh kanan dan separuh kiri paling banyak satu.

[2] Gambar-gambar di bab ini diambil dari *Problem Solving with Algorithms and Data Structures Using Python*, oleh Bradley Miller dan David Ranum, penerbit Franklin, Beedle & Associates, 2011. Tersedia versi online-nya di `http://interactivepython.org/runestone/static/pythonds/index.html`




**Gambar** 6.1: Membelah list sampai tiap sub-list berisi satu elemen atau kosong. Sesudah itu digabung seperti ditunjukkan di Gambar 6.2.




**Gambar** 6.2: Menggabungkan list satu demi satu.

```
1  def mergeSort(A):
2      #print("Membelah     ", A)
3      if len(A) > 1:
4          mid = len(A) // 2       # Membelah list.
5          separuhKiri = A[:mid]   # Slicing ini langkah yang expensive sebenarnya,
6          separuhKanan = A[mid:]  #   bisakah kamu membuatnya lebih baik?
7
8          mergeSort(separuhKiri)  # Ini rekursi. Memanggil lebih lanjut mergeSort
9          mergeSort(separuhKanan) #   untuk separuhKiri dan separuhKanan.
10
11         # Di bawah ini adalah proses penggabungan.
12         i=0  ;  j=0  ;  k=0
13         while i < len(separuhKiri) and j < len(separuhKanan):
14             if separuhKiri[i] < separuhKanan[j]:  # while-loop ini
15                 A[k] = separuhKiri[i]             #   menggabungkan kedua list, yakni
16                 i = i + 1                         #   separuhKiri dan separuhKanan,
17             else:                                 #   sampai salah satu kosong.
18                 A[k] = separuhKanan[j]            # Perhatikan kesamaan strukturnya
19                 j = j + 1                         #   dengan proses penggabungan
20             k=k+1                                 #   dua list urut.
21
22         while i < len(separuhKiri):   # Jika separuhKiri mempunyai sisa
23             A[k] = separuhKiri[i]     #   tumpukkan ke A
24             i = i + 1                 #   satu demi satu.
25             k = k + 1                 #
26
27         while j < len(separuhKanan):  # Jika separuhKanan mempunyai sisa
28             A[k] = separuhKanan[j]    #   tumpukkan ke A
29             j = j + 1                 #   satu demi satu.
30             k = k + 1
31     #print("Menggabungkan", A)
```

Larikan program di atas dengan memanggilnya seperti ini

```
alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

Setelah fungsi mergeSort ini dipanggil pada separuh kiri dan separuh kanan (baris 8–9), kita anggap list itu semua sudah diurutkan[3]. Baris-baris selanjutnya (baris 12–30) bertugas menggabungkan dua list yang lebih kecil untuk menjadi list yang lebih besar. Perhatikan bahwa operasi penggabungan menempatkan kembali elemen-elemen ke list yang asli satu demi satu dengan secara berulang mengambil elemen terkecil dari list yang sudah urut[4].

Fungsi `mergeSort` di atas telah dilengkapi sebuah perintah `print` (baris 2) untuk menunjukkan isi list yang sedang diurutkan di awal tiap pemanggilan. Terdapat pula sebuah perintah `print` (baris 31) untuk memperlihatkan proses penggabungan. Aktifkan baris-baris itu[5] dan jalankan programnya. Hasilnya kurang lebih adalah sebagai berikut.

[3] Ingat, ini adalah fungsi rekursif.

[4] Inilah kegunaan index $k$ di program itu. Lihat bahwa struktur program penggabungan ini (baris 12–30) sama dengan program kita di halaman 56, kecuali bahwa di sini kita tidak membuat list baru dan ada index $k$. List yang kita diminta mengurutkan adalah juga list tempat kita menaruh hasil pengurutannya.

[5] Yakni, hilangkan tanda # di baris-baris itu. Jangan lupa untuk memberi kembali tanda itu saat tes kecepatan nanti.

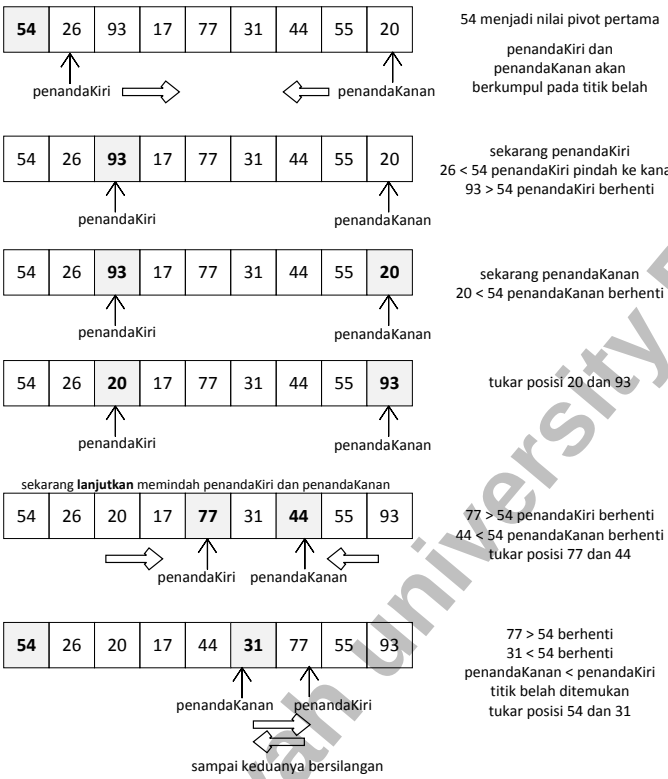
```
Membelah      [54, 26, 93, 17, 77, 31, 44, 55, 20]
Membelah      [54, 26, 93, 17]
Membelah      [54, 26]
Membelah      [54]
Menggabungkan [54]
Membelah      [26]
Menggabungkan [26]
Menggabungkan [26, 54]
Membelah      [93, 17]
Membelah      [93]
Menggabungkan [93]
Membelah      [17]
Menggabungkan [17]
Menggabungkan [17, 93]
Menggabungkan [17, 26, 54, 93]
Membelah      [77, 31, 44, 55, 20]
Membelah      [77, 31]
Membelah      [77]
Menggabungkan [77]
Membelah      [31]
Menggabungkan [31]
Menggabungkan [31, 77]
Membelah      [44, 55, 20]
Membelah      [44]
Menggabungkan [44]
Membelah      [55, 20]
Membelah      [55]
Menggabungkan [55]
Membelah      [20]
Menggabungkan [20]
Menggabungkan [20, 55]
Menggabungkan [20, 44, 55]
Menggabungkan [20, 31, 44, 55, 77]
Menggabungkan [17, 20, 26, 31, 44, 54, 55, 77, 93]
```

Bisa dilihat bahwa proses pembelahan pada akhirnya menghasilkan sebuah list yang bisa langsung digabungkan dengan list lain yang urut.

**Latihan 6.1** Memakai bolpen merah atau biru, tandai dan beri nomer urut eksekusi proses pada Gambar 6.1 dan 6.2, dengan mengacu pada output di atas[6]. □

Merge sort ini secara umum bekerja lebih cepat dari insertion sort ataupun selection sort. (Dapatkah kamu melihat mengapa demikian?) Namun program di atas masih dapat ditingkatkan efisiensinya. Di program itu kita masih menggunakan operator *slice* (seperti `A[:mid]` dan `A[mid:]`), yang cukup memakan waktu. Kita meningkatkan efisiensi program dengan tidak memakai operator slice ini, dan lalu mem-*pass* index awal dan index akhir bersama list-nya saat kita memanggilnya secara rekursif. *Kamu dapat mengerjakannya sebagai latihan. Kamu perlu memisah fungsi* `mergeSort` *di atas menjadi beberapa fungsi, mirip halnya dengan apa yang dilakukan algoritma Quick Sort di bawah.*

[6] Jika kamu membaca modul ini dari file pdf-nya, maka cetaklah halaman-halaman yang relevan, atau kamu dapat menulis ulang dua gambar itu.




**Gambar** 6.3: ***Quick sort.*** Mencari titik belah untuk 54. Tujuan langkah ini adalah membuat semua elemen yang lebih kecil dari 54 berada di sebelah kiri dan semua elemen yang lebih besar dari 54 berada di sebelah kanan (meskipun di dalamnya sendiri belum urut). Angka 54 sendiri akan sudah berada di tempat yang tepat. Ini bisa dilihat di gambar selanjutnya, yaitu Gambar 6.4.

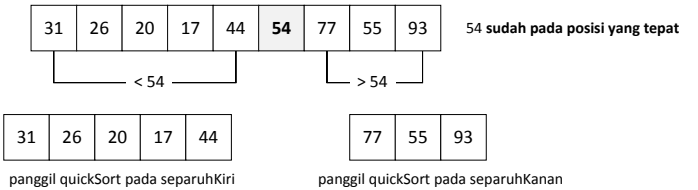
## 6.3 Quick sort

Algoritma quick sort memakai 'belah dan taklukkan' untuk mendapatkan keuntungan yang sama dengan merge sort, dengan tidak memerlukan penyimpanan tambahan. Harga yang harus dibayar adalah bahwa ada kemungkinan list itu tidak dibelah sama panjang.

Sebuah quicksort pertama-tama memilih sebuah nilai di antara nilai-nilai yang ada di list, yang disebut **nilai pivot**. Banyak cara untuk memilih nilai pivot itu[7]. Kita saat ini memilih nilai yang paling kiri (berarti elemen pertama). Peran nilai pivot ini adalah untuk membantu dalam proses membelah list ini.

Sesudah itu kita membuat semua elemen yang lebih kecil dari pivot berada di sebelah kiri dan semua elemen yang lebih besar dari pivot berada di sebelah kanan, serta pivotnya sendiri berada di antar dua sub-list ini. Pada contoh di Gambar 6.3, elemen pivot kita adalah $\boxed{54}$.

[7]Misal: yang paling kiri, yang paling kanan, secara acak, atau median-dari-tiga




**Gambar** 6.4: Keadaan setelah Gambar 6.3. Sekarang semua elemen di sebelah kiri 54 adalah lebih kecil dari 54, dan semua elemen di sebelah kanan 54 adalah lebih besar dari 54 (meskipun di masing-masingnya sendiri belum urut). Dengan demikian ***54 sudah menempati posisi yang tepat di list itu***. Lalu, secara rekursif kedua paruh kiri dan kanan kembali di-*feed* ke algoritma quick sort.

Kita mulai dengan menaikkan penandaKiri satu demi satu sampai kita menemui sebuah nilai yang lebih besar dari pivot. Kita lalu menurunkan penandaKanan sampai kita menemukan sebuah nilai yang lebih kecil dari pivot. Pada saat ini kita telah menemukan dua item yang posisinya salah (dilihat dari titik belahnya). Untuk contoh yang sedang kita bahas di Gambar 6.3, ini adalah 93 dan 20. Sekarang kita dapat menukar posisi dua elemen ini dan prosesnya diulang kembali.

Pada saat di mana penandaKanan menjadi lebih kecil dari penandaKiri, kita berhenti. Posisi penandaKanan sekarang adalah titik belahnya. Nilai pivot dapat ditukar posisinya dengan isi titik belahnya dan sekarang nilai pivotnya sudah berada pada posisi yang tepat (Gambar 6.4). Semua item di sebelah titik belah adalah lebih kecil dari nilai pivotnya dan semua item di sebelah kanan adalah lebih besar dari nilai pivotnya. List ini sekarang dapat dibagi dua pada titik belah itu dan lalu algoritma quick sort dapat kembali dipanggil secara rekursif pada kedua belahan itu.

Fungsi `quickSort` pada kode di bawah ini memanggil fungsi rekursif `quickSortBantu`. Fungsi `quickSortBantu` mulai dengan base case yang sama dengan merge sort (lihat kode `mergeSort` di halaman 59). Jika panjang list-nya kurang dari atau sama dengan satu, berarti sudah terurut. Jika lebih besar dari satu, maka list itu dapat dipartisi dan diurutkan secara rekursif. Fungsi `partisi` mengimplementasikan proses yang dijelaskan sebelumnya.

Fungsi `quickSort` adalah interface ke luar:

```
1 def quickSort(A):
2     quickSortBantu(A, 0, len(A) - 1 ) # memanggil quickSortBantu di
3
```

Fungsi `quickSortBantu` adalah fungsi rekursif yang dipanggil berulang-ulang:

```
4 def quickSortBantu(A, awal, akhir):
5     if awal < akhir:
6         titikBelah = partisi(A, awal, akhir)      # Atur elemen dan dapatkan titikBelah.
7         quickSortBantu(A, awal, titikBelah - 1)  # Ini rekursi untuk belah sisi kiri
8         quickSortBantu(A, titikBelah + 1, akhir) #   dan belah sisi kanan.
9
```

Fungsi `partisi` adalah untuk mencari titik belah dan mengatur ulang posisi elemen-elemen

agar menyesuaikan terhadap `nilaiPivot`:

```
10 def partisi(A, awal, akhir):
11     nilaiPivot = A[awal] # Di sini nilaiPivot kita ambil dari elemen yang paling kiri.
12
13     penandaKiri = awal + 1 # Posisi awal penandaKiri. Lihat Gambar 6.3.
14     penandaKanan = akhir   # Posisi awal penandaKanan.
15
16     selesai = False
17     while not selesai: # loop di bawah adalah untuk mengatur ulang posisi semua elemen
18
19         while penandaKiri <= penandaKanan and \   # penandaKiri bergerak ke kanan,
20                 A[penandaKiri] <= nilaiPivot:     #  sampai ketemu suatu nilai yang
21             penandaKiri = penandaKiri + 1         #  lebih besar dari nilaiPivot
22
23         while A[penandaKanan] >= nilaiPivot and \ # penandaKanan bergerak ke kiri,
24                 penandaKanan >= penandaKiri:      #  sampai ketemu suatu nilai yang
25             penandaKanan = penandaKanan - 1       #  lebih kecil dari nilaiPivot
26
27         if penandaKanan < penandaKiri:            # Kalau dua penanda sudah bersilangan,
28             selesai = True                        #  selesai & lanjut ke penempatan pivot
29         else:
30             temp = A[penandaKiri]                 # Tapi kalau belum bersilangan,
31             A[penandaKiri] = A[penandaKanan]      #  tukarlah isi yang ditunjuk oleh
32             A[penandaKanan] = temp                #  penandaKiri dan penandaKanan
33
34     temp = A[awal]               # Kalau acara tukar menukar posisi sudah selesai,
35     A[awal] = A[penandaKanan]    #  kita lalu menempatkan pivot pada posisi yang tepat,
36     A[penandaKanan] = temp       #  yakni posisi penandaKanan. Lihat Gambar 6.3 dan 6.4.
37                                  #  Posisi penandaKanan adalah juga titikBelah.
38     return penandaKanan     # Fungsi ini mengembalikan titikBelah ke pemanggil
```

Sebelumnya telah kita bahas bahwa pemilihan pivot bisa memakai berbagai cara. Khususnya, kita bisa mencoba menghindari pembelahan yang berpotensi tidak seimbang dengan menggunakan teknik median-dari-tiga. Untuk memilih nilai pivot, kita melihat elemen pertama, tengah, dan terakhir[8]. Untuk contoh kita di atas ini adalah 54, 77, 20. Dari ketiganya, pilih median-nya. Dalam contoh kita ini berarti 54. (Kebetulan ini juga yang kita pilih sebagai pivot awal di contoh kita.)

Ide dasarnya adalah ketika ada suatu kejadian di mana elemen paling kiri ternyata nilainya 'tidak cukup tengah', metode median-dari-tiga ini akan lebih baik dalam memilihkan 'nilai tengah'. Ini bisa berguna saat list awalnya sudah agak urut. Sebagai latihan, *ubahlah kode di atas untuk mengaplikasikan pemilihan pivot memakai metode median-dari-tiga.*

## 6.4 Soal-soal untuk Mahasiswa

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

1. Ubahlah kode `mergeSort` dan `quickSort` di atas agar bisa mengurutkan list yang berisi object-object `mhsTIF` yang sudah kamu buat di Modul 2. Uji programmu secukupnya.

[8] Atau, dengan kata lain, elemen paling kiri, tengah, dan paling kanan.

2. Memakai bolpen merah atau biru, tandai dan beri nomer urut eksekusi proses pada Gambar 6.1 dan 6.2, dengan mengacu pada output di halaman 59.

3. Uji kecepatan. Ujilah `mergeSort` dan `quickSort` di atas (bersama metode sort yang kamu pelajari sebelumnya) dengan kode di bawah ini.

```
from time import time as detak
from random import shuffle as kocok
import time
k = range(6000)
kocok(k)
u_bub = k[:] ##
u_sel = k[:] ##  Deep copy.
u_ins = k[:] ##  Jangan lupa [:]-nya!
u_mrg = k[:] ##
u_qck = k[:] ##

aw=detak();bubbleSort(u_bub);ak=detak();print('bubble: %g detik' %(ak-aw) );
aw=detak();selectionSort(u_sel);ak=detak();print('selection: %g detik' %(ak-aw) );
aw=detak();insertionSort(u_ins);ak=detak();print('insertion: %g detik' %(ak-aw) );
aw=detak();mergeSort(u_mrg);ak=detak();print('merge: %g detik' %(ak-aw) );
aw=detak();quickSort(u_qck);ak=detak();print('quick: %g detik' %(ak-aw) );
```

   Tunjukkan hasil ujinya ke asisten praktikum[9].

4. Diberikan list `L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]`, gambarlah *trace* pengurutan[10] untuk algoritma

   (a) merge sort

   (b) quick sort

***Soal-soal di bawah ini sedikit lebih sulit.*** Kerjakanlah di rumah.

5. Tingkatkan efisiensi program `mergeSort` dengan *tidak* memakai operator slice (seperti `A[:mid]` dan `A[mid:]`), dan lalu mem-*pass* index awal dan index akhir bersama listnya saat kita memanggil `mergeSort` secara rekursif. Kamu akan perlu memisah fungsi `mergeSort` itu menjadi beberapa fungsi, mirip halnya dengan apa yang dilakukan algoritma quick sort.

6. Apakah kita bisa meningkatkan efisiensi program `quickSort` dengan memakai metode median-dari-tiga untuk memilih pivotnya? Ubahlah kodenya dan ujilah.

7. Uji-kecepatan keduanya dan perbandingkan juga dengan kode awalnya.

8. Buatlah versi linked-list untuk program `mergeSort` di atas.

[9] Ekspektasi hasil: `mergeSort` dan `quickSort` hampir sama cepat, keduanya sekitar 20 kali lebih cepat dari `selectionSort` dan `insertionSort`, dan sekitar 100 kali lebih cepat dari `bubbleSort`.

[10] Kurang lebih sama dengan yang dimaksud soal nomer 2 di atas.

# Modul 7

# *Regular Expressions*

*Regular expression*, sering disebut *regex*, adalah suatu bahasa yang handal untuk mencocokkan pola di suatu string. Praktikum ini akan memperkenalkan peserta pada fasilitas-fasilitas regex yang ada pada Python[1]. Modul Python yang memiliki fasilitas ini adalah modul `re`. Mulailah memuat modul itu ke memori dengan perintah

```
>>> import re
```

Dalam Python, regex biasa ditulis dalam bentuk

```
cocok = re.findall(pola, sebuahString)
```

Metode `re.findall()` di atas menerima sebuah pola regex dan sebuah string, lalu mencari rangkaian karakter yang sesuai dengan pola itu di string yang dimaksud. Jika pencariannya berhasil, `findall()` mengembalikan sebuah *list* berisi semua string yang ditemukan dan mengembalikan list kosong kalau tidak berhasil. Contoh berikut akan mencari pola `'kata:'` diikuti sebuah kata 3 huruf. Ketiklah lalu jalankan.

```
1 s = 'sebuah contoh kata:teh!!'
2 cocok = re.findall(r'kata:\w\w\w', s)
3 # Pernyataan-IF sesudah findall() akan memeriksa apakah pencarian berhasil:
4 if cocok:
5     print('menemukan', cocok ) ## 'menemukan [kata:teh]'
6 else:
7     print('tidak menemukan')
```

Kode `cocok = re.findall(pola, s)`, seperti pada baris 2 di atas, menyimpan hasil pencarian pada sebuah variabel bernama "cocok". Sesudah itu pernyataan-IF akan memeriksa `cocok` – jika berisi sesuatu maka pencariannya berhasil dan `cocok` adalah list teks-teks yang cocok (dalam contoh di atas adalah satu buah `'kata:teh'`). Selain itu jika cocok-nya kosong, maka berarti tidak ada teks yang cocok dengan polanya.

Huruf 'r' pada permulaan teks pola menunjukkan teks "mentah" yang melewatkan karakter *backslash* (yakni "\") tanpa perubahan. Kamu disarankan agar selalu memakai 'r' ini ketika menulis regex.

[1]Di sini kita tidak akan belajar bagaimana *membuat* mesin regex-nya. Itu merupakan topik lanjut.

## 7.1 Pola-Pola Dasar

Kekuatan regex adalah bahwa dia bisa membuat pola-pola khusus, bukan hanya karakter yang *fixed*. Berikut ini adalah pola-pola paling dasar yang akan cocok dengan sebuah atau beberapa karakter.

Jika saat pertama kali membaca kamu tidak paham maksud pola-pola di bawah ini, jangan putus asa! Kerjakan saja dulu latihan-latihan di modul ini, sedikit demi sedikit, lalu nanti kembali ke sini untuk referensi.

- `a, X, 9` — karakter-karakter biasa akan cocok dengan tepat. Karakter khusus yang tidak dengan sendirinya cocok, karena punya makna khusus, adalah `.^$*+?{}[]\|()` (karakter ini harus di-*escape* kalau kita kebetulan memang ingin secara khusus menangkap karakter itu. Detail di bawah.)
- `.` (sebuah titik) — akan cocok dengan satu karakter apapun kecual garis baru `\n`
- `^=` awal, `$=` akhir — cocok dengan awal dan akhir suatu string.
- `*` — akan mencocokkan 0 atau lebih kemunculan pola di sebelah kirinya. Misal, `ab*` akan menangkap `'a'`, `'ab'`, atau `'a'` diikuti huruf `'b'` sebanyak apapun.
- `+` — akan mencocokkan 1 atau lebih kemunculan pola di sebelah kirinya. Misal, `ab+` akan cocok dengan `'ab'`, atau `'abb'` (atau sebanyak apapun `b`-nya), tapi TIDAK cocok dengan `'a'` saja.
- `?` — akan mencocokkan 0 atau 1 kemunculan pola di sebelah kirinya. Misal, `ab?` akan cocok dengan `'a'` atau `'ab'` saja.
- `*?`, `+?`, `??` — Ini adalah versi "tidak-rakus" dari `*`, `+`, `?`. Tidak-rakus maksudnya dia akan *sesedikit mungkin* meraup ke kanan.
- `{m}` — menentukan bahwa ada $m$ perulangan dari pola sebelah kirinya yang cocok. Misal `a{6}` akan cocok dengan enam karakter `'a'`, tapi tidak lima.
- `{m,n}` — mencocokkan sebanyak $m$ sampai $n$ pengulangan pola di sebelah kirinya. Sebagai contoh, `a{3,5}` akan cocok dengan 3 sampai 5 buah karakter `'a'`. Jika $m$ tidak ditulis, maka batas bawahnya dianggap nol, dan jika $n$ tidak ditulis, batas atasnya dianggap tak terhingga. Sebagai contoh, pola `a{4,}b` akan cocok dengan `aaaab` atau seribu karakter `'a'` diikuti sebuah `b`. Tapi `aaab` tidak akan cocok (perhatikan bahwa `'a'`-nya cuma 3).
- `[ ]` — kurung siku digunakan untuk mengindikasikan himpunan karekter. Pada sebuah himpunan karakter:
  - Karakter-karakter dapat ditulis satu-persatu, misal pola `[ums]` akan cocok dengan `'u'`, `'m'`, atau `'s'`.
  - Jangkauan/range karakter diindikasikan dengan menulis dua karakter dan dipisahkan dengan `'-'`, misal pola `[a-z]` akan cocok dengan semua huruf kecil ASCII, pola `[0-5][0-9]` akan cocok dengan semua bilangan dua-digit dari `00` sampai `59`, dan

pola `[0-9A-Fa-f]` akan cocok dengan angka hexadecimal manapun. Jika `'-'` di-escape (misal `[a\-z]`) atau ditaruh sebagai karakter pertama atau terakhir (misal `[a-]`), dia akan cocok dengan karakter literal `'-'`.

  - Karkater khusus *kehilangan makna khususnya* di dalam himpunan. Sebagai contoh, pola `[(+*)]` akan cocok dengan yang manapun saja dari karakter literal `'('`, `'+'`, `'*'`, atau `')'`.
  - Kelas karakter seperti `\w` atau `\s` (dijelaskan di bawah) diterima di dalam himpunan.
  - Karakter yang di luar jangkauan dapat dibuat cocok dengan membuat komplemen *"complement"* himpunan itu. Jika karakter pertama di sebuah himpunan adalah `'^'`, maka semua karakter yang *tidak* di himpunan itu akan cocok. Sebagai contoh, pola `[^5]` akan cocok dengan semua karakter kecuali `'5'`. Karakter `'^'` tidak memiliki makna khusus jika tidak ditaruh di posisi pertama sebuah himpunan.

- `|` — Pola `A|B`, di mana `A` dan `B` adalah pola regex apapun, akan membuat regex baru yang mencocokkan pola `A` atau `B`. Jumlah regex yang bisa di-atau-kan ini bisa banyak. Ini bisa dilakukan juga di dalam group (lihat di bawah). Saat string target dipindai ("di-scan"), regex-regex yang dipisahkan oleh `'|'` akan dites dari kiri ke kanan. Ketika satu pola cocok, maka cabang itu diterima. Ini berarti, jika `A` cocok, maka `B` tidak akan diperiksa lagi, meski sebenarnya bisa menghasilkan *match* yang lebih panjang.

- `\w` — huruf w kecil akan cocok dengan karakter kata biasa ("word"): sebuah huruf atau angka atau garis bawah `[a-zA-Z0-9_]`. Perhatikan bahwa meskipun kita boleh menghapalkannya dengan kata "word", pola `\w` hanya akan cocok dengan *satu buah* karakter, tidak seluruh kata. Dan ingat juga bahwa `\W` (dengan huruf W besar) akan cocok dengan karakter non-kata (misal tanda bintang `*`, tanda ampersand `&`).

- `\b` — batas antara kata dan non-kata.

- `\s` — huruf s kecil akan cocok dengan satu karakter putih – spasi, baris baru, 'Enter', tab, dan bentuk `[\n\r\t\f]`. Pola `\S` (huruf S besar) akan cocok dengan karakter non putih.

- `\t, \n, \r` — tab, baris baru, dan 'Enter'

- `\d` — angka `[0-9]`

- `\` — mencegah "kekhususan" suatu karakter; inilah yang dimaksud dengan meng-escape. Misalnya, gunakan `\.` untuk mencocokkan sebuah karakter titik. Gunakan `\\` untuk mencocokkan karakter backslash.

Untuk referensi yang lebih lengkap, silakan kunjungi docs.python.org/3/library/re.html

**Latihan 7.1** Jalankan kode di atas (halaman 65), pastikan hasilnya adalah seperti yang kamu duga. Di sana, kita memberi perintah seperti ini: "di dalam `string` temukan teks yang memuat `'kata:'` yang diikuti tiga huruf". Perhatikan bahwa ini dicapai dengan menyetel pola = `r'kata:\w\w\w'`. Sekarang jawablah yang berikut:

- Apa yang terjadi kalau kodenya dirubah di baris pertamanya? Seperti ini:
  `string = 'sebuah contoh kata:batagor!!'`
- Bagaimana kalau
  `string = 'sebuah contoh kata:es teh!!'`

□

## 7.2 Contoh-contoh Dasar

Aturan dasar pencarian pola oleh regex di dalam suatu string adalah:

- Pencarian berjalan di suatu string dari awal sampai akhir.
- Semua pola harus cocok, tapi tidak seluruh string perlu cocok dengan polanya.
- Jika `cocok = re.findall(pola, sebuahString)` berhasil, maka `cocok` berisi daftar teks yang cocok.

```
## Dua baris ini mencari pola 'eee' di string 'teeeh'.
##   Seluruh pola harus cocok, tapi itu bisa muncul di mana saja.
##   Jika berhasil, \texttt{cocok} adalah daftar semua teks yang cocok.
cocok = re.findall(r'eee', 'teeeh') #=>  cocok == ['eee']
cocok = re.findall(r'ehs', 'teeeh') #=>  cocok == []

## . = semua karakter kecuali \n
cocok = re.findall(r'..h', 'teeeh') #=>  cocok == ['eeh']

## \d = karakter angka, \w = karakter huruf atau angka
cocok = re.findall(r'\d\d\d', 't123h di 2019 bulan 02') #=> cocok == ['123', '201']
cocok = re.findall(r'\w\w\w', '@@a*bc#def*tghh!!') #=> cocok == ['def', 'tgh']
```

**Latihan 7.2** Pada kode di atas tambahkan baris-baris untuk mengeluarkan hasilnya[2]. Sesudah itu jalankan programnya dan pastikan hasilnya seperti yang kamu duga sebelumnya. Pastikan juga semua pertanyaan "kok bisa seperti itu ya" yang muncul di pikiranmu terjawab tuntas. Tanyalah kepada dosen kuliah atau dosen praktikum jika kamu belum paham akan suatu konsep di latihan ini. □

## 7.3 Pengulangan dan Kurung Siku

Memprogram regex menjadi lebih menarik saat kita memakai `+` dan `*` untuk membentuk pengulangan (*repetition*) di polanya.

- `+` — 1 atau lebih kemunculan pola di sebelah kirinya. Contoh `'e+'` berarti "satu atau lebih `e`"
- `*` — 0 atau lebih kemunculan pola di sebelah kirinya.
- `?` — cocok dengan 0 atau 1 kemunculan pola di sebelah kirinya.

Ingat, ingat, ada tiga pola kemunculan: 1-atau-lebih, 0-atau-lebih, 0-atau-1.

[2] Misal `print(cocok.group())`

### Paling kiri dan paling besar

Pertama-tama pencarian akan menemukan kecocokan yang paling kiri untuk polanya, dan, yang kedua, pencarian ini akan mencoba meraup sebanyak-banyaknya string. Dengan kata lain, `+` dan `*` ini akan meraup sejauh-jauhnya (`+` dan `*` ini sering disebut "rakus"). Jika diinginkan agar regex-nya "tidak rakus", gunakan `+?` dan `*?`.

## 7.3.1 Contoh pengulangan

```
1  ## e+ = satu atau lebih e, sebanyak-banyaknya.
2  cocok = re.findall(r'te+', 'ghdteeeh') #=>  cocok == ['teee']
3
4  ## Menemukan solusi yang paling kiri, dan dari situ mendorong si tanda '+'
5  ##     sejauh-jauhnya (ingat 'paling kiri dan paling besar').
6  ## Pada contoh ini, perhatikan bahwa pencarian menemukan dua pola yang tepat.
7  cocok = re.findall(r'e+', 'teeheeee') #=> cocok == ['ee', 'eeee']
8
9  ## \s* = nol atau lebih karakter putih (spasi, tab, dsb.)
10 ## Di sini mencari 3 angka, kemungkinan dipisahkan oleh spasi/tab.
11 polanya = r'\d\s*\d\s*\d'
12 cocok = re.findall(polanya, 'xx1 2   3xx') #=>  cocok == ['1 2   3']
13 cocok = re.findall(polanya, 'xx12  3xx')   #=>  cocok == ['12  3']
14 cocok = re.findall(polanya, 'xx123xx')     #=>  cocok == ['123']
15
16 ## ^ -> cocok dengan awal string, jadi ini tidak akan menemukan:
17 cocok = re.findall(r'^k\w+', 'mejakursi') #=>  tidak ketemu, cocok == []
18 ## tapi tanpa ^ dia berhasil:
19 cocok = re.findall(r'k[\w\s]+', 'mejakursi tamu saya')#=> cocok == ['kursi tamu saya']
```

**Latihan 7.3** Pada kode di atas tambahkan baris-baris untuk mengeluarkan hasilnya. Sesudah itu jalankan programnya dan pastikan hasilnya seperti contoh. Berikutnya, buatlah analisis cara kerja regex-nya pada baris 7, 12, dan 19. □

Sekarang, bisakah kita meng-ekstrak alamat email dari pengetahuan di atas? Mari kita coba dengan memakai `'\w+'` yang barusan kita pelajari. (Silakan dijalankan!)

```
s = 'Alamatku adalah dita-b@google.com mas'
cocok = re.findall(r'\w+@\w+', s)
print(cocok[0])          ## =>   'b@google'
```

Ternyata ada yang kelewatan! Simbol '`-`' dan '`.`' tidak ditangkap oleh polanya. Di sini kita memerlukan kurung siku.

## 7.3.2 Kurung Siku

Seperti dijelaskan di atas, kurung siku (karakter '`[`' dan '`]`') dapat dipakai untuk mengindikasikan *sekelompok* karakter, jadi `[abc]` akan cocok dengan `'a'`, `'b'`, atau `'c'`. Kode seperti `\w`, `\s` dan sebagainya tetap berlaku di dalam kurung siku dengan pengecualian simbol titik (`.`). Dia bermakna titik biasa, begitu saja. Titik. Untuk problem ekstraksi alamat email kita di atas, kurung siku merupakan cara mudah untuk menambahkan `'.'` dan `'-'` pada himpunan karakter yang muncul di sekitaran simbol @ dengan pola `[\w.-]+@[\w.-]+` untuk mendapatkan alamat email yang utuh:

```
cocok = re.findall(r'[\w.-]+@[\w.-]+', s)
print(cocok[0])      ## => 'dita-b@google.com'
```

**Latihan 7.4** Pola di atas sudah bisa mengekstrak email dengan cukup bagus. Namun masih ada yang kurang. Nah, pola berikut bisa mengekstrak dengan lebih baik (meski tetap saja belum sempurna). Buatlah analisisnya!

```
pola = r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+'
```

Peningkatan apa yang diberikan oleh pola baru ini dibandingkan dengan pola sebelumnya? □

## 7.4 Ekstraksi secara Group

Fasilitas *grouping* di regex memungkinkan kita untuk memilah-milah bagian teks yang telah cocok. Misal pada problem alamat email di atas kita ingin mengekstrak *username* dan *host*-nya secara terpisah. Untuk melakukan hal ini, perlu kita tambahkan tanda kurung `()` di seputar username dan host di polanya, seperti ini: `r'([\w.-]+)@([\w.-]+)'`.

Di sini, tandakurungnya tidak mengubah pola yang akan cocok, tapi dia akan mengelompokkan teks-teks yang cocok. Jika pencarian berhasil (lihat contoh di bawah), object `cocok` akan berisi tuple-tuple yang berisi semua temuan. Tiap-tiap tuple akan berisi pasangan sesuai pola `grouping`-nya.

Di contoh berikut ini, perhatikan bahwa `cocok[0]` adalah sebuah tuple. Lalu `cocok[0][0]` adalah teks cocok yang terkait pasangan tanda kurung pertama, dan `cocok[0][1]` adalah teks cocok yang terkait pasangan tanda kurung kedua. Jadi, setiap tuple mempunyai panjang 2 dan berisi username dan host, misal `('sri','google.com')`.

```
s = 'Alamatku adalah dita-b@google.com mas'
cocok = re.findall(r'([\w.-]+)@([\w.-]+)', s) ## perhatikan posisi () di polanya
cocok  ##  adalah [('dita-b', 'google.com')]
## Bisa kita pilah satu per satu seperti ini:
cocok[0][0]  ## 'dita-b'
cocok[0][1]  ## 'google.com'
```

Alur kerja yang biasa dilakukan adalah pertama-tama kita tulis regex untuk teks yang kita cari, lalu menambahkan kelompok tandakurung untuk mengekstrak bagian-bagian yang kita inginkan.

Sekarang cobalah yang berikut.

```
## Kita punya banyak alamat email
s = 'Alamatku sri@google.com serta joko@abc.com ok bro. atau don@email.com'

## Di sini re.findall() mengembalikan sebuah list beranggotakan string alamat
pola = r'[\w\.-]+@[\w\.-]+'
e = re.findall(pola, s)
print(e)
##=> e akan berisi ['sri@google.com', 'joko@abc.com', 'don@email.com']
```

Sekarang kita siap untuk meng-group polanya menjadi username dan host.

```
pola = r'([\w\.-]+)@([\w\.-]+)'
e = re.findall(pola, s)
print(e)     ##==> sekarang bagaimanakah hasilnya?
## Atau kita cetak satu per satu:
for tup in e:
    print('user', tup[0], 'dengan host:', tup[1])
```

**Latihan 7.5** Tambahkan beberapa kata dan alamat email pada string `s` di atas. Jalankan, dan pastikan alamat email yang baru kamu sertakan akan tertangkap dan 'terbelah' oleh regex-nya. Buatlah analisisnya. □

### 7.4.1 Pencarian dalam berkas

Untuk berkas, mungkin kamu punya kebiasaan menulis sebuah loop yang memutari baris demi baris, lalu memanggil `findall()` pada tiap baris. TAPI, ada yang lebih baik! Berikan semuanya pada `findall()` dan dia akan mengembalikan *semua* teks yang cocok. Berikut ini adalah ilustrasinya.

```
f = open('test.txt', 'r', encoding='latin1') ## membuka file.
teks = f.read()
f.close()
p = r'sebuah pola'        ## ini polanya.
## memberikan seluruhnya ke findall()
## dia mengembalikan list beranggotakan string yang cocok
strings = re.findall(p, teks)
```

Beberapa latihan didedikasikan untuk ini. Silakan dikerjakan.

## 7.5 Referensi lebih lanjut

Regex adalah topik yang luas. Untuk belajar lebih lanjut, silakan buka

- docs.python.org/3/library/re.html
- docs.python.org/3/howto/regex.html
- developers.google.com/edu/python/regular-expressions
- www.tutorialspoint.com/python/python_reg_expressions.htm

## 7.6 Soal-soal untuk Mahasiswa

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

1. **Meng-ekstrak kata-kata dengan awalan 'me'**. Bukalah halaman web id.wikipedia.org/wiki/Indonesia. Salin seluruh tubuh artikel itu dan tuang ke Notepad[3]. Simpan sebagai `Indonesia.txt`. Tugasmu adalah mengekstrak semua kata yang mempunyai awalan

[3]Memakai `Ctrl-A`, lalu `Ctrl-C`, lalu `Ctrl-V` juga boleh.

'me', lalu menyimpannya dalam sebuah tuple. Perhatikan bahwa 'me' bisa muncul di awal, tengah ataupun akhir kalimat[4].

2. **Meng-ekstrak kata-kata dengan awalan 'di'**. Sama seperti di atas, tapi yang kamu ekstrak adalah kata-kata dengan awalan 'di'. Perhatikan, yang dimaksud di sini adalah *awalan* 'di' (menunjukkan kata kerja pasif), bukan *kata depan* 'di' (menunjukkan keterangan tempat), yang kita bahas di nomer di bawah.

3. **Meng-ekstrak kata depan 'di' dan tempat yang ditunjuknya**. Di sini tugasmu adalah mengekstrak – masih dari berkas `Indonesia.txt` – kata depan 'di' *beserta satu kata yang mengikutinya*. Misal, dari string

   ```
   'Saya dilahirkan dan dibesarkan di sini, di Indonesia, sebuah negeri indah
   permai di mana para saudagar di masa lampau berdatangan.'
   ```

   akan ter-ekstrak `('di sini', 'di Indonesia', 'di mana', 'di masa')`

4. **Meng-ekstrak data dari berkas `.html`**. Tugasmu di sini adalah membuat sebuah *list of tuples* yang memuat nama-nama negara beserta *Innovation index*-nya. Kunjungilah halaman en.wikipedia.org/wiki/Knowledge_Economic_Index. Pastikan halaman itu memuat sebuah tabel besar hasil penelitian. Kerjakan langkah-langkah berikut.

   - Simpanlah halaman itu sebagai berkas html murni, dengan nama `KEI.html`[5].
   - Bukalah berkas `KEI.html` itu dengan Notepad++ atau Notepad. Perhatikan segmen-segmen di file itu yang mempunyai isi seperti di bawah

     ```
     ...
      </span><a href="/wiki/Belgium" title="Belgium">Belgium</a></td>
     <td>8.73</td>
     <td>8.70</td>
     <td>8.82</td>
     <td>8.96</td>
     <td>9.14</td>
     <td>8.02</td>
     <td>16</td>
     </tr>
     <tr>
     ...
     ```

   - Akses file itu menggunakan `f=open('KEI.html', 'r', encoding='latin1')` dan baca file menggunakan `teks = f.read()`. Lalu tutup aksesnya menggunakan `f.close()`.
   - Sebagai awalan, ekstraklah *semua* nama-nama negara yang ada di berkas itu memakai regex, dengan *memperhatikan pola di sekitaran nama-nama negara*[6].

[4] Perhatikan pula bahwa regex-mu juga akan menangkap kata-kata seperti 'meskipun', 'mereka', 'mesin', 'Meulaboh', dan 'Merauke'. Mereka ini tentu saja bukan kata kerja. (Nomer di bawahnya juga mempunyai masalah serupa.) Untuk sekarang, tidak mengapa kalau kata-kata itu tertangkap. Bisakah kamu memikirkan suatu cara agar kata-kata itu diabaikan oleh regex-mu?

[5] Caranya bisa dengan meng-klik kanan di halaman itu lalu klik "Save as..." (untuk Google Chrome), atau "Save Page As..." (untuk Mozilla Firefox). Ketika muncul sebuah *dialog box*, simpan di folder kerjamu sebagai KEI.html dan – di bawahnya – pilih "Web page, HTML Only".

[6] Lihatlah bahwa kamu mempunyai tiga pilihan untuk meng-ekstrak nama negara itu:

- Sekarang ekstraklah kolom "Innovation" untuk semua negara di tabel itu (untuk contoh Belgium di atas, kolom ini adalah kolom yang angkanya 8.96 ) memakai regex[7]. Sebagai tambahan, ubahlah string `'8.96'` menjadi float.
- Dengan konsep group (memakai tanda kurung `(...)` di tempat yang tepat, lihat Section 7.4 di halaman 70), ekstraklah nama negara beserta *Innovation Index*-nya, lalu buatlah kode seperlunya untuk memodifikasinya, sehingga akhirnya kamu mempunyai *list of tuples* seperti berikut

```
[('Belgium', 8.96), ('Malaysia', 6.83), ('Indonesia', 3.32), ...]
```

**Penting:** di bab ini kita membahas telah regular expression untuk mem-*parse* teks dengan pola tertentu. Di bagian latihan, kita mencoba mem-`parse` file html untuk diekstrak informasinya. Namun sebenarnya *regular expression* tidaklah handal untuk mem-*parse* file html.

Terdapat modul python khusus untuk *parsing* suatu file html, namanya *Beautiful Soup*. Jika kamu akan melakukan *html parsing* ataupun *web scraping*, gunakan *Beautiful Soup* ini.



– apakah memakai pola `/wiki/Negara`,
– atau pola `title="Negara"`,
– atau pola `">Negara</a></td>`.

Salah satunya "lebih benar" dari yang lainnya.

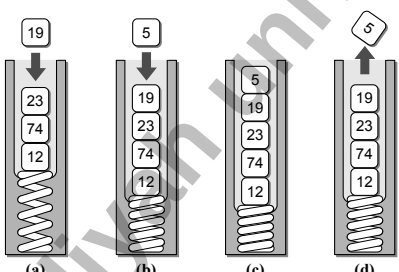
[7] Kira-kira bagaimanakah strateginya? Petunjuk kecil: perhatikan bahwa angka 8.96 di atas tercantum empat `<td>` sesudah `</a></td>`. Dan, bagaimanakah kamu membuat pola "satu angka diikuti titik diikuti dua angka"?

# Modul 8

# *Stacks and Queues*

Di modul ini kita akan belajar struktur *stack* (tumpukan) dan *queue* (antrian).

## 8.1 Pengertian *Stack*




**Gambar** 8.1: Tampilan abstrak sebuah stack: (a) mendorong (*push*) angka 19; (b) mendorong angka 5; (c) hasil setelah angka 19 dan 5 didorong; dan (d) mengeluarkan (*pop*) angka paling atas.

*Stack*, diterjemahkan menjadi *tumpukan*, digunakan untuk menyimpan data sedemikian hingga item terakhir yang disisipkan adalah item pertama. Stack digunakan untuk mengimplementasikan protokol *last-in-first-out* (LIFO). Stack adalah sebuah struktur data linear di mana item baru ditambahkan dan diambil dari ujung yang sama, yakni puncak tumpukan (***top** of the stack*). Di bagian paling bawah adalah *base* atau dasar-nya.

Konsep stack sangat umum dalam literatur ilmu komputer dan banyak digunakan di berbagai penyelesaian masalah. Stack juga ditemui dalam kehidupan sehari-hari[1]. Perhatikan gambaran abstrak stack di Gambar 8.1[2].

Dalam definisi yang lebih formal, sebuah stack adalah sebuah struktur data yang menyimpan koleksi linear yang mempunyai akses terbatas pada urutan terakhir-masuk-keluar-pertama.

[1]Seperti: tumpukan piring, tumpukan buku di kardus, atau parkir mobil dengan satu gerbang saja di mana mobil yang terakhir masuk adalah yang pertama keluar.

[2]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.

Menambah dan mengambil item dibatasi pada satu ujung saja, yakni *puncak* tumpukan (*top* of the stack). Stack kosong (*empty stack*) adalah stack yang tidak mempunyai item.

## 8.2 *Features* dan *properties* sebuah stack

Bayangkan kamu mempunyai sebuah kardus kosong *Indomie*® dan akan menyimpan buku-buku ukuran A4 dengan posisi tidur di situ. Apa saja yang bisa kita tanyakan, dan kita lakukan, pada tumpukan itu? Setidaknya:

1. Apakah kosong?
2. Apakah penuh?
3. Berapa banyakkah buku di tumpukan itu?
4. Taruh satu buku di tumpukan itu
5. Ambil satu buku dari tumpukan itu
6. Intip, buku apakah yang paling atas

Dari *requirements* ini kita bisa mulai merancang beberapa perintah yang harus ada di class Stack yang akan dibuat.

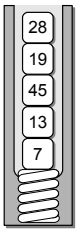
- `Stack()`: membuat stack baru yang kosong
- `isEmpty()`: mengembalikan nilai boolean yang menunjukkan apakah stack itu kosong.
- `length()`: mengembalikan banyaknya item di dalam stack
- `pop()`: mengembalikan nilai dari item yang paling atas dan menghapusnya, jika stack-nya tidak kosong.
- `peek()`: mengembalikan nilai dari item yang paling atas, tanpa menghapusnya.
- `push( item )`: menambah `item` ke puncak stack.

Bisa kamu lihat bahwa *untuk saat ini* kita tidak mengimplementasikan perintah yang berasosiasi dengan "Apakah penuh". Jika diperlukan hal ini bisa dibuat, namun secara default kita bisa menumpuk sebanyak-banyaknya ke tumpukan yang dibentuk dengan list Python[3].

Berikut ini adalah contoh pemanggilannya.

```
PROMPT = "Masukkan bilangan positif (<0 untuk mengakhiri):"
myStack = Stack()               # Membuat stack baru (program di halaman berikut)
value = int(input( PROMPT ))
while value >= 0 :              # Memasukkan satu per satu
    myStack.push( value )
    value = int(input( PROMPT ))
while not myStack.isEmpty() :   # Mengeluarkan satu per satu
    value = myStack.pop()
    print( value )
```

[3]Jika kamu memakai bahasa C, kamu bisa menumpuk sebanyak-banyaknya di suatu stack dengan memakai struktur data linked-list. Sesungguhnya list di bahasa Python diimplementasikan dengan doubly-linked-list di bahasa C (yakni bahasa yang digunakan untuk membuat bahasa Python)




**Gambar** 8.2: Hasil stack setelah mengeksekusi contoh.

Ketika user memasukkan (*push*), secara berurutan, nilai 7, 13, 45, 19, 28, -1, maka kita mempunyai stack yang sepeti ditunjukkan di Gambar 8.2. Perhatikan bahwa kita mempunyai nilai yang paling dasar adalah nilai yang pertama kita masukkan, dan yang paling atas adalah yang paling terakhir kita masukkan. Kalau kita mengeluarkan (*pop*), maka urutannya adalah dari yang paling atas. Lihat Gambar 8.2[4].

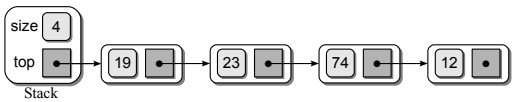
## 8.3 Implementasi Stack

Stack dapat diimplementasikan dengan beberapa cara. Dua yang paling umum adalah memakai list dan linked list.

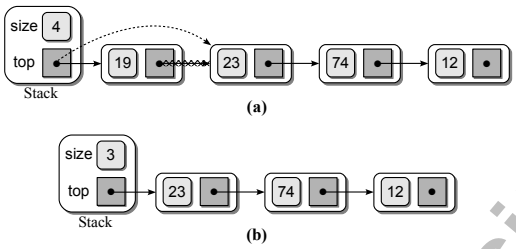
### Menggunakan list

Berikut ini implementasi stack menggunakan list Python.

```
class  Stack(object):
    def __init__(self):          # Membuat stack kosong.
        self.items = []          # List untuk menyimpan stack.

    def isEmpty(self):           # Mengembalikan True kalau kosong,
        return len(self)==0      #   selain itu False

    def __len__(self):           # Mengembalikan banyaknya item di stack.
        return len(self.items) #

    def peek(self):              # Mengembalikan nilai posisi atas tanpa menghapus.
        assert not self.isEmpty(), "Stack kosong. Tidak bisa diintip"
        return self.items[-1]

    def pop(self):               # Mengembalikan nilai posisi atas lalu menghapus.
        assert not self.isEmpty(), "Stack kosong. Tidak bisa di-pop"
        return self.items.pop()

    def push(self, data):        # Mendorong item baru ke stack.
        self.items.append(data)
```




**Gambar** 8.3: Contoh object stack yang diimplementasikan dengan linked-list.




**Gambar** 8.4: Menge-pop sebuah item dari stack. (a) modifikasi link yang diperlukan, (b) hasil setelah mengeluarkan item yang paling atas.

### Menggunakan linked-list

Berikut ini adalah implementasi stack menggunakan linked-list.

```
1  class StackLL(object):
2
3      def __init__(self):
4          self.top = None
5          self.size = 0
6
7      def isEmpty(self):
8          return self.top is None
9
10     def __len__(self):
11         return self.size
12
13     def peek(self):
14         assert not self.isEmpty(), "Tidak bisa diintip. Stack kosong."
15         return self.top.item
16
17     def pop(self):
18         assert not self.isEmpty(), "Tidak bisa pop dari stack kosong."
19         node = self.top
20         self.top = self.top.next
21         self.size -= 1
22         return node.item
23
24     def push(self, data):
25         self.top =  _StackNode(data, self.top)
26         self.size += 1
27
28 class _StackNode(object):           # Ini adalah kelas privat
29     def __init__(self, data, link): #  untuk menyimpan data. Dia
30         self.item = data            #  dipanggil oleh class StackLL
```

[4]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.

```
31          self.next = link            #  di atas.
```

Perhatikan bahwa materi linked-list-nya dibuat dengan class yang berbeda, yakni `_StackNode`. Gambar 8.3 dan 8.4 memperlihatkan contoh operasinya.

## 8.4 Contoh program

Stack digunakan secara sangat luas dalam *computing*. Interupsi program dan sifat multi-tasking komputer modern bergantung salah satunya pada konsep stack ini. Kita belum akan menyentuh materi itu di matakuliah ini, namun kamu bisa 'merasakan' mekanisme algoritmanya melalui contoh-contoh program.

### Mengubah bilangan desimal ke biner

Program di bawah ini mengubah representasi suatu bilangan, dari basis sepuluh ke basis dua, yang penjelasannya telah kamu dapatkan di kelas.

```
1  def cetakBiner(d):
2      f = Stack()          # Atau: f =  StackLL()
3      if d==0: f.push(0);
4      while d !=0:
5          sisa = d%2
6          d = d//2
7          f.push(sisa)
8      st = ""
9      for i in range(len(f)):
10         st = st + str(f.pop())
11     return st
```

Berikut ini adalah contoh pemanggilan dan keluarannya.

```
>>> cetakBiner(11)
'1011'
>>> cetakBiner(53)
'110101'
```

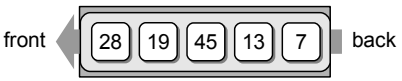
## 8.5 Pengertian *Queue*

*Queue*[5] atau antrian adalah sebuah struktur data yang memodelkan fenomena antrian. Seperti sudah kamu duga, antrian memiliki sifat "yang duluan datang adalah yang duluan keluar", *First in First out*, disingkat FIFO. Lihat Gambar 8.5. Apa saja yang bisa kita *query* ke sebuah antrian?

- Buatlah sebuah antrian. Ini akan diimplementasikan dengan class `Queue()`.
- Apakah antriannya kosong? Ini akan kita diimplementasikan dengan metode `isEmpty()`.
- Berapa panjangkah antriannya? Ini akan diimplementasikan dengan metode `__len__()`.

[5] Dibaca [kyu]

- Masukkan item *xyz* ini ke antrian. Ini akan diimplementasikan dengan metode `enqueue(data)`.
- Ambil item *pqr* ini dari antrian. Ini akan diimplementasikan dengan metode `dequeue()`.




**Gambar** 8.5: Tampilan abstrak sebuah sebuah antrian. Bagian depan (front) adalah yang terlebih dulu datang, bagian belakang (back) adalah yang datang belakangan. Pada antrian yang biasa, yang terlebih dulu datang akan keluar terlebih dahulu.

Dari sini bisa kita tulis kode untuk membuat antrian seperti diilustrasikan pada Gambar 8.5 di atas seperti di bawah ini.

```
Q = Queue()
Q.enqueue( 28 )
Q.enqueue( 19 )
Q.enqueue( 45 )
Q.enqueue( 13 )
Q.enqueue( 7 )
```

Setelah membuat object baru, kita lalu mengantrikan lima angka dengan urutan seperti urutan tampilnya angka-angka itu di antrian. Kita lalu bisa mengambil angka atau menambahkan angka ke antrian itu. Gambar 8.6 mengilustrasikan beberapa operasi tambahan pada contoh antrian di atas.

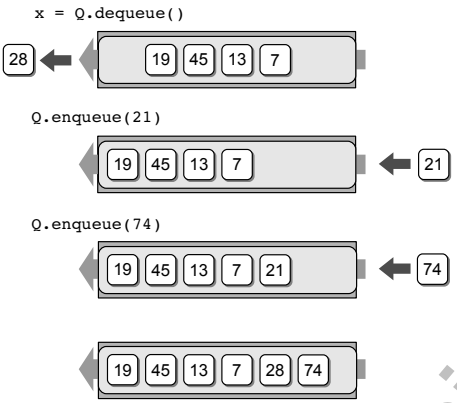
## 8.6 Implementasi

Implementasi antrian paling sederhana adalah memakai list di Python, seperti kode di bawah ini.

```
1  class Queue(object):
2      def __init__(self):
3          self.qlist = []
4
5      def isEmpty(self):
6          return len(self) == 0
7
8      def __len__(self):
9          return len(self.qlist)
10
11     def enqueue( self, data ):
12         self.qlist.append(data)
13
14     def dequeue(self):
15         assert not self.isEmpty(), "Antrian sedang kosong."
16         return self.qlist.pop(0)
```

Cobalah beberapa perintah untuk memeriksa validitas program di atas.

```
## Ini melanjutkan kode di atas
Q.dequeue()  # Perhatikan urutan hasilnya.
Q.dequeue()  #  Apakah sesuai dengan urutan masuknya?
Q.dequeue()
```




**Gambar** 8.6: Sebuah antrian saat kepadanya dilakukan beberapa operasi tambahan. Mengambil 28 dari antrian, mengantrikan 21, lalu mengantrikan 74.

```
Q.dequeue()
Q.dequeue()
Q.dequeue()  # Apakah ada pesan error?
Q.enqueue(98)
Q.enqueue(54)
Q.dequeue()
```

## 8.7 *Priority Queues*

Priority queue adalah antrian dengan prioritas. Berikut ini adalah perbedaan antara antrian biasa dengan antrian dengan prioritas:

- Ketika suatu item akan dimasukkan ke antrian (`enqueue`), item ini disertai dengan suatu penanda prioritas. Ini biasanya berupa bilangan bulat dari 0 sampai suatu bilangan tertentu yang dipilih sesuai aplikasi (misalnya kita memilih 4, yang berarti ada 5 prioritas dengan 0 sebagai prioritas tertinggi).
- Ketika suatu item akan diambil dari antrian (`dequeue`), maka yang diambil adalah yang prioritasnya tertinggi, dengan tidak memperdulikan urutan kedatangan. Jadi misal kita mempunyai lima item di suatu antrian dengan semuanya mempunyai prioritas 4, dan lalu datang item baru (`enqueue`) dengan prioritas 2, maka ketika "dipanggil" (dengan `dequeue`), yang keluar terlebih dahulu adalah item dengan prioritas 2 tadi.
- Jika ada dua item atau lebih dengan prioritas yang sama, yang didahulukan keluar adalah yang terlebih dahulu masuk. Prinsip FIFO tetap berlaku pada item-item dengan prioritas sama.

Berikut ini contoh pemanggilannya.

```
S = PriorityQueue()
S.enqueue("Jeruk", 4)
S.enqueue("Tomat", 2)
```

```
S.enqueue("Mangga", 0)
S.enqueue("Duku", 5)
S.enqueue("Pepaya", 2)
S.dequeue()          # Akan mengeluarkan "Mangga" karena prioritasnya tertinggi.
S.dequeue()          # Akan mengeluarkan "Tomat".
S.dequeue()          # Kalau ini "Pepaya".
```

Untuk mengimplementasikan *priority queue* ini, kita membuat suatu kelas baru yang dipakai untuk menyatukan item dengan prioritas yang dilekatkan padanya. Berikut ini sebagian dari implementasi *priority queue* ini.

```
class PriorityQueue(object):

    def __init__(self):
        self.qlist = []

    def __len__(self):
        return len(self.qlist)

    def isEmpty(self):
        return len(self) == 0

    def enqueue(self, data, priority):
        entry = _PriorityQEntry( item, priority ) # Memanggil object _PriorityQEntry
        self.qlist.append(entry)

    def dequeue(self):
        pass # .... lihat Soal-soal untuk Mahasiswa.
```

Berikut ini adalah class untuk menyimpan item beserta prioritasnya.

```
class _PriorityQEntry(object):
    def __init__(self, data, priority):
        self.item = data
        self.priority = priority
```

## 8.8 Soal-soal untuk Mahasiswa

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

### *Stacks*

1. Buatlah program untuk mengubah representasi suatu bilangan dari basis sepuluh ke basis dua. Berikut ini contoh pemanggilannya.

```
>>> cetakHexa(12)
'C'
>>> cetakHexa(31)
'1F'
>>> cetakHexa(229)
'E5'
>>> cetakHexa(255)
'FF'
>>> cetakHexa(31519)
```

```
'7B1F'
```

Perhatikan bahwa sisa pembagian tidak hanya 0 dan 1, namun bisa 0 sampai 9 dan bahkan 10, 11, 12, 13, 14, 15. Kamu harus memetakan angka-angka yang lebih dari 9 ke lambang A, B, C, D, E, dan F.

2. Eksekusi program berikut dengan pensil dan kertas, dan tunjukkan isi stack-nya pada setiap langkah.

```
nilai = Stack()
for i in range(16 ):
    if i % 3 == 0:
        nilai.push( i )
```

3. Eksekusi program berikut dengan pensil dan kertas, dan tunjukkan isi stack-nya pada setiap langkah.

```
nilai = Stack()
for i in range( 16 ) :
    if i % 3 == 0 :
        nilai.push( i )
    elif i % 4 == 0 :
        nilai.pop()
```

### *Queues*

4. Tulis dua metode berikut ke class `Queue` dan class `PriorityQueue` di atas

   - Metode untuk mengetahui item yang paling depan tanpa menghapusnya

     ```
     def getFrontMost(self):
         ## Tulis perintahnya di sini
     ```

   - Metode untuk mengetahui item yang paling belakang tanpa menghapusnya

     ```
     def getRearMost(self):
         ## Tulis perintahnya di sini
     ```

5. Pada class `PriorityQueue` di atas, metode `dequeue()` belum diimplementasikan. Tulislah metode dequeue() ini dengan memperhatikan syarat-syarat seperti yang telah dicantumkan di halaman 81.

# Modul 9

# Pohon Biner

Selama ini kita telah mempelajari struktur dalam bentuk *sequential*, yakni sebuah struktur data di mana sebuah elemen berada "sebelum" atau "sesudah" elemen lain. Namun terkadang kita memerlukan bentuk yang tidak linier. Nah, di modul ini kita akan mempelajari sebuah struktur hirarkis yang kita namai "pohon".

## 9.1 Pengantar dan contoh

Pohon atau *Tree* adalah sebuah struktur data yang menyimpan elemen-elemen informasi dalam bentuk hirarkis. Struktur ini terdiri dari *nodes* (simpul) dan *edges* (garis penghubung). Antar simpul mempunyai hubungan "ortu," "anak," "leluhur," "keturunan," dan sebagainya. Data disimpan dalam ***simpul***[1] dan pasangan simpul dihubungkan dengan sebuah ***pinggir***[2].

Pinggir-pinggir ini mewakili relasi antar simpul yang terhubungkan dengan panah/garis untuk membentuk struktur hirarkis yang menyerupai pohon terbalik dengan cabang, daun, dan akar.

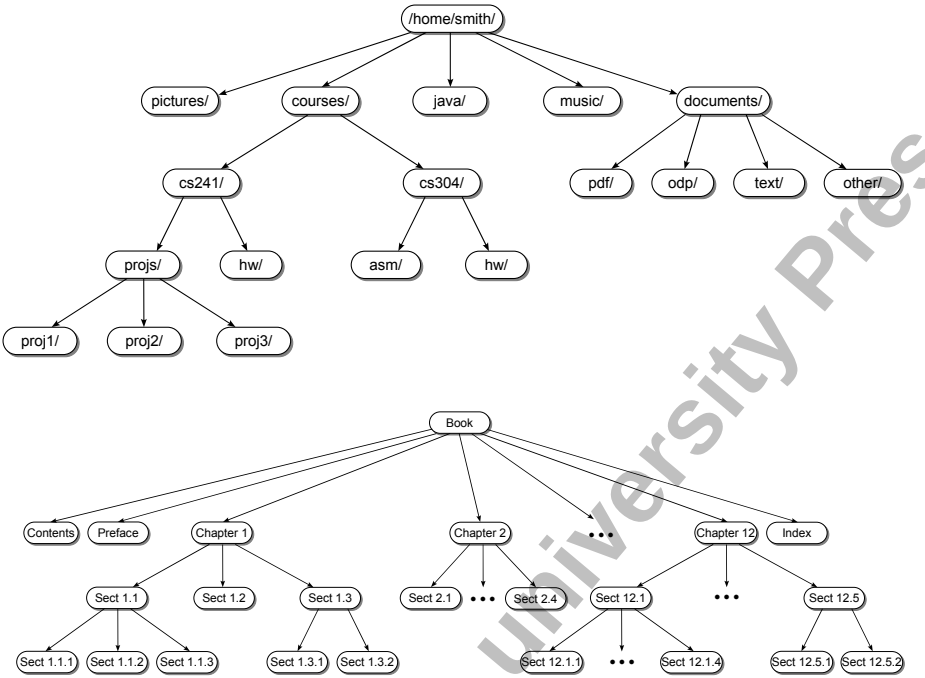
Sebuah contoh struktur pohon adalah direktori dan sub-direktori di sistem berkas pada harddisk komputer. Pohon bisa dipakai untuk merepresentasikan data yang terstruktur, yang mana akan menghasilkan pembagian data menjadi data yang lebih kecil dan lebih kecil lagi. Contohnya adalah isi sebuah buku yang dibagi menjadi bab, section, dan seterusnya. Gambar 9.1 menunjukkan dua contoh di atas[3].

**Latihan 9.1** Sebuah contoh pohon yang lain adalah struktur pengalamatan domain di web. Misal domain `.id` berperan sebagai akarnya. Maka di bawahnya ada sub-domain seperti `go.id`, `co.id`, `ac.id`, `net.id`, `sch.id`. Dan, contoh berikutnya, di bawah `ac.id` ada domain seperti `ugm.ac.id`, `uns.ac.id`, `ui.ac.id`, `itb.ac.id`, `ums.ac.id`. Di bawah `ums.ac.id` ada (di antaranya) `pmb.ums.ac.id`, `www.ums.ac.id`, `krsonline.ums.ac.id`, `star.ums.ac.id`, `fki.ums.ac.id`. Sekarang gambarlah struktur pohon domain-domain ini seingat kamu. □

[1] Bahasa Inggrisnya: ***node***

[2] Bahasa Inggrisnya: ***edge***

[3] Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.




**Gambar** 9.1: Contoh-contoh struktur pohon. Atas: direktori dan sub-direktori di sebuah komputer dengan sistem operasi LINUX. Bawah: pembagian isi sebuah buku.

Pada bab ini kita akan mempelajari struktur pohon yang lebih khusus, yakni **pohon biner** (*binary tree*). Pohon biner adalah pohon di mana setiap simpul dapat mempunyai anak paling banyak dua[4].

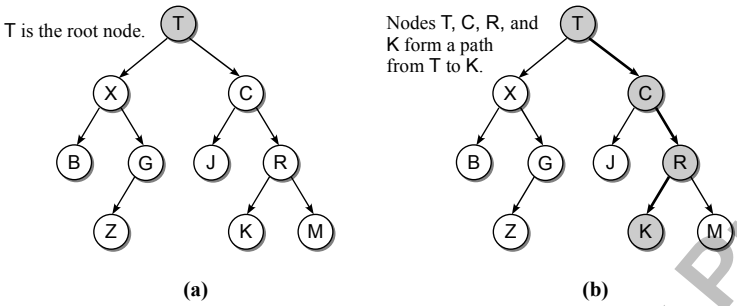
## 9.2 Istilah-istilah

Kita akan menggunakan berbagai istilah yang sebaiknya kamu ketahui maknanya sejak sekarang.

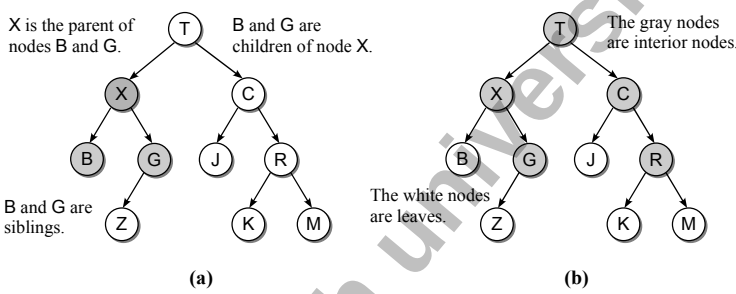
### Akar

**Simpul akar** (dari frase bahasa inggris *root node*) adalah simpul yang paling atas. Simpul akar ini adalah simpul yang tidak mempunyai pinggir yang datang. Lihat contoh pada Gambar 9.2(a) di mana akar pohonnya adalah `T`.

[4]Ini bukan program pemerintah RI lewat BKKBN :-) . Kita mempelajari pohon biner terlebih dahulu karena lebih sederhana operasinya. Jika kamu sudah menguasai operasi pada pohon biner, maka memahami operasi pada pohon yang lebih umum akan meng-extend konsepnya saja




**Gambar** 9.2: Contoh pohon biner. (a) pohon dengan akar `T`. (b) pohon dengan jalur dari `T` ke `K` yang dibentuk dari simpul-simpul `T-C-R-K`.




**Gambar** 9.3: Contoh pohon biner dengan: (a) hubungan ortu, anak, dan saudara; dan (b) perbedaan antara simpul interior dan simpul daun.

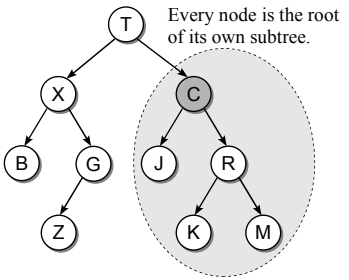
### Jalur

Setelah di simpul akar, simpul-simpul lain diakses melalui pinggir-pinggir. Dimulai dari akar, berjalan mengikuti arah panah sampai simpul yang dituju dicapai. Simpul-simpul yang dikunjungi saat meniti pinggir-pinggir, dari simpul awal sampai simpul tujuan, membentuk suatu ***jalur*** (diterjemahkan bebas dari kata Bahasa Inggris *path*). Seperti ditunjukkan di Gambar 9.2(b), simpul-simpul yang berlabel `T`, `C`, `R`, dan `K` membentuk jalur dari `T` ke `K`.

### Ortu

Setiap simpul, kecuali simpul akar, mempunyai satu ***simpul ortu*** (dari frase Bahasa Inggris *parent node*), yang diidentifikasi dari pinggir yang datang. Sebuah simpul hanya dapat mempunyai satu ortu (yakni, hanya satu pinggir datang). Ini menjadikan adanya jalur yang unik dari simpul akar ke simpul-simpul lain yang manapun di pohon itu[5]. Pada Gambar 9.3(a), simpul `X` adalah ortu dari simpul `B` dan `G`.

[5]Maksudnya, hanya ada satu jalur yang bisa dibuat dari akar ke simpul lain. Contoh: pada Gambar 9.2(b), *satu-satunya* jalur dari simpul akar ke simpul `K` adalah `T-C-R-K`. Tidak ada yang lain.




**Gambar** 9.4: Sebuah subpohon dengan simpul akar `C`.

### Anak

Setiap simpul pada pohon biner dapat mempunyai satu atau dua ***anak***, yang membuat hirarki ortu-anak. Anak sebuah simpul diidentifikasi dari pinggir yang mengarah ke luar. Sebagai contoh, simpul `B` dan `G` keduanya adalah anak dari simpul `X`. Semua simpul yang mempunyai ortu yang sama dinamakan ***saudara*** (diterjemahkan bebas dari kata Bahasa Inggris *sibling*), tapi tidak ada akses langsung antar saudara. Sehingga kita tidak bisa secara langsung mengakses simpul `C` dari simpul `X` dan sebaliknya.

### Simpul Interior dan Simpul Daun

Simpul yang memiliki sekurangnya satu anak disebut ***simpul interior*** atau simpul dalam. Simpul yang tidak mempunyai anak disebut ***simpul daun***. Pada Gambar 9.3(b), simpul interior diberi latar belakang abu-abu dan simpul daun diberi latar belakang warna putih.

### Subpohon

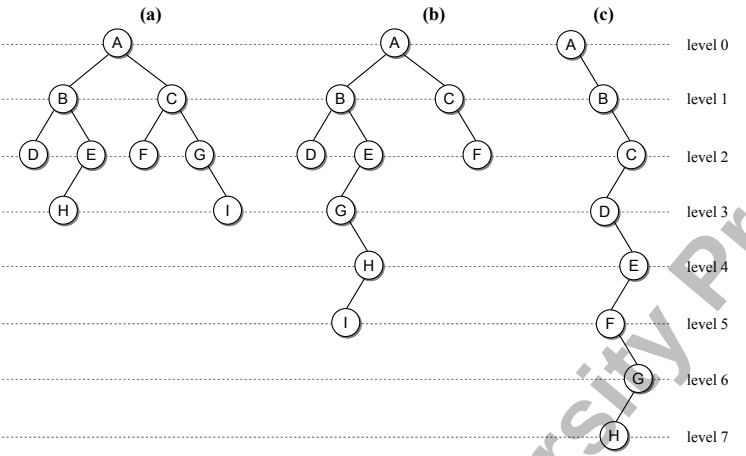
Dilihat dari pengertiannya, sebuah pohon adalah struktur rekursif. Setiap simpul dapat menjadi akar dari ***subpohon*** (subtree)-nya sendiri, yang terdiri atas simpul-simpul dan pinggir-pinggir dari pohon utamanya yang lebih besar. Gambar 9.4 menunjukkan subpohon dengan `C` sebagai akarnya.

### Kerabat

Semua simpul pada sebuah subpohon adalah ***keturunan*** dari simpul akar subpohon itu. Pada pohon contoh di Gambar 9.4, simpul `J`, `R`, `K`, dan `M` adalah keturunan dari simpul `C`. ***Leluhur*** sebuah simpul adalah ortu, kakek, buyut, dan seterusnya naik sampai ke simpul akar. Leluhur sebuah simpul dapat juga diidentifikasi dari simpul-simpul sepanjang jalur dari akar ke simpul itu.

## 9.3 Properti Pohon Biner

Sebuah pohon biner adalah sebuah pohon di mana setiap simpul dapat mempunyai anak paling banyak dua. Satu simpul anak dinamai ***anak kiri*** dan satunya lagi dinamai ***anak kanan***. Di




**Gambar** 9.5: Ada berbagai bentuk pohon untuk jumlah simpul yang sama.

sini kita akan meninjau beberapa hal terkait pohon biner.

Pohon biner mempunyai beragam bentuk dan ukuran. Bentuknya bermacam-macam bergantung pada jumlah simpul dan bagaimana simpul-simpulnya dihubungkan, lihat Gambar 9.5. Terdapat bermacam properti dan karakter yang diasosiasikan dengan pohon biner, yang semuanya bergantung pada pengaturan letak simpul-simpul di dalam pohon itu.

### Ukuran Pohon

Simpul-simpul di sebuah pohon biner diorganisasi ke dalam beberapa ***level*** dengan simpul akar di level 0, anaknya di simpul 1, anak dari anaknya di simpul 2, dan seterusnya. Dalam istilah pohon keluarga, satu level adalah satu generasi. Pohon biner di Gambar 9.5(a), sebagai contoh, mempunyai dua simpul di level satu (`B` dan `C`), empat simpul di level dua (`D`, `E`, `F`, dan `G`), serta dua simpul di level tiga (`H` dan `I`). Simpul akar selalu menempati level nol.

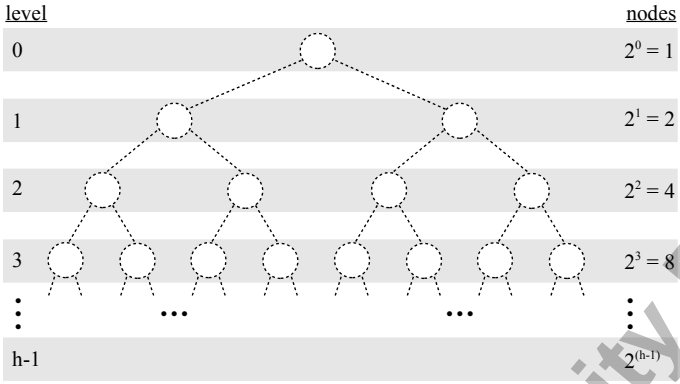
***Kedalaman*** *(depth)* sebuah simpul adalah jarak dia dari simpul akar, di mana jarak adalah banyaknya level yang memisahkan keduanya. Keadalaman sebuah simpul berkaitan dengan level di mana dia berada. Misal simpul `G` di Gambar 9.5. Di pohon (a), `G` mempunyai kedalaman 2, di pohon (b) dia memiliki kedalaman 3, dan di (c) kedalamannya adalah 6.

***Ketinggian*** *(height)* sebuah pohon biner adalah jumlah level di pohon itu[6]. Sebagai contoh, pada Gambar 9.5, pohon (a) mempunyai ketinggian 4.

***Lebar*** *(width)* suatu pohon biner adalah jumlah simpul pada level yang mempunyai simpul terbanyak. Pada tiga pohon biner di Gambar 9.5, pohon (a) mempunyai lebar 4, pohon (b) mempunyai lebar 3, dan pohon (c) mempunyai lebar 1.

***Besar*** atau ***ukuran*** *(size)* suatu pohon biner adalah, mudah saja, banyaknya simpul di pohon

[6]Jangan mengacaukan kedalaman dan ketinggian! Kedalaman mengacu pada posisi simpul di pohon biner itu, sedangkan ketinggian mengacu pada tinggi pohon biner secara keseluruhan




**Gambar** 9.6: Slot-slot yang mungkin untuk penempatan simpul-simpul pada sebuah pohon biner.

itu. Sebuah pohon yang kosong mempunyai ketinggial 0, lebar 0, dan besar 0.

Sebuah pohon biner dengan ukuran $n$ dapat mempunyai ketinggian maksimum $n$, yang akan terjadi jika terdapat satu simpul di setiap level (lihat pohon biner pada Gambar 9.5a). Sekarang, berapakah ketinggian minimumnya? Untuk menentukan ini, kita harus mempertimbangkan jumlah simpul maksimum di tiap level, karena simpul-simpul itu semua harus ditata di tiap level dengan kapasitas penuh[7].Gambar 9.6 mengilustrasikan slot-slot penempatan yang mungkin di sebuah pohon biner.

Karena tiap simpul dapat mempunyai anak paling banyak dua, setiap level berikutnya akan mempunyai simpul dua kali lebih banyak dari simpul sebelumnya. Sehingga level ke $i$ mempunyai kapasitas sebanyak $2^i$ simpul. Jika kita menjumlahkan ukuran setiap level, saat semua level-level ini telah penuh sesuai kapasitas (kecuali mungkin level terakhirnya), kita akan menemukan bahwa ketinggian minimum sebuah pohon biner dengan ukuran $n$ adalah $\lfloor \log_2 n \rfloor + 1$. Di sini, lambang $\lfloor \cdot rfloor + 1$ adalah operasi pembulatan ke bawah.

**Latihan 9.2** Hitunglah ketinggian minimum sebuah pohon biner dengan ukuran berikut dan buatlah sketsa pohon biner untuk masing-masing ukuran.

a. 2 c. 5 e. 8 g. 12 i. 20

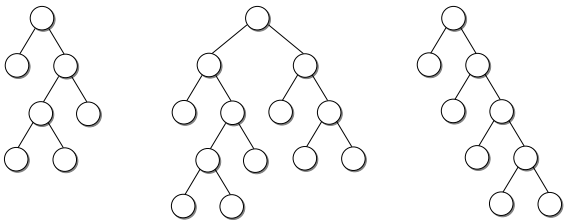
b. 3 d. 7 f. 11 h. 15 j. 31

□

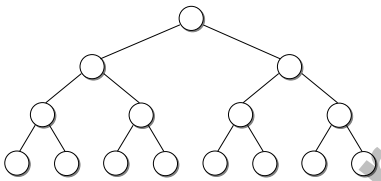
### Struktur Pohon

Ketinggian pohon berperan penting dalam menganalisis kompleksitas-waktu dari beragam algoritma yang dioperasikan ke pohon biner. Sifat struktural pohon biner dapat pula berperan dalam efisiensi suatu algoritma. Bahkan beberapa algoritma mensyaratkan struktur tertentu.

[7]Level 0 dipenuhi terlebih dahulu, lalu level 1 dipenuhi, lalu level 2 dipenuhi, dst.



**Gambar** 9.7: Beberapa contoh pohon biner penuh.



**Gambar** 9.8: Sebuah pohon biner sempurna.

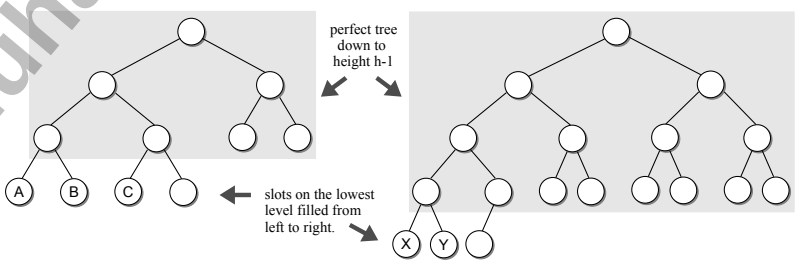
Sebuah ***pohon biner penuh*** *(full binary tree)* adalah sebuah pohon biner di mana setiap simpul dalam berisi dua anak. Pohon penuh bisa mempunyai bentuk yang bermacam-macam, seperti diperlihatkan di Gambar 9.7.

Sebuah***pohon biner sempurna*** *(perfect binary tree)* adalah sebuah pohon biner penuh di mana semua simpul daun berada di level yang sama. Pohon sempurna ini mempunyai slot-slot yang penuh dari atas sampai bawah tanpa ada yang kosong, seperti diperlihatkan di Gambar 9.8.

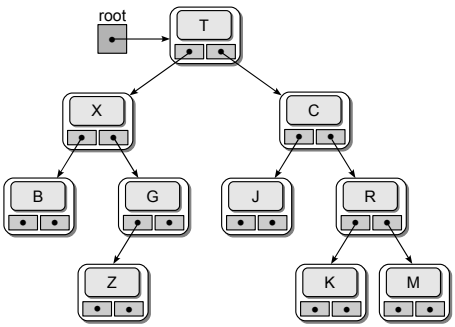
Sebuah pohon biner dengan ketinggian $h$ disebut ***pohon biner komplit*** *(complete binary tree)* jika dia merupakan pohon biner sempurna sampai tinggi $h-1$ dan semua simpul-simpul di level terakhir mengisi slot-slot yang tersedia dari kiri ke kanan tanpa ada *gap*. Contoh pohon biner komplit ditunjukkan di Gambar 9.9.

**Latihan 9.3** Kerjakan yang berikut ini.

a. Adakah kekangan terkait ukuran sebuah pohon biner jika kita ingin membuat sebuah pohon biner penuh? Bisakah 4 simpul menjadi sebuah pohon biner? Bagaimana kalau 5 simpul?




**Gambar** 9.9: Dua contoh pohon biner komplit.




**Gambar** 9.10: Implementasi sebuah pohon biner.

b. Ukuran sebuah pohon biner sempurna adalah tertentu; tidak setiap jumlah simpul bisa disusun menjadi pohon biner sempurna. Buktikan bahwa sejumlah simpul bisa disusun menjadi sebuah pohon biner sempurna jika, dan hanya jika, banyaknya simpul itu adalah $2^n - 1$ dengan $n = 1, 2, 3, \ldots$.

□

## 9.4 Implementasi

Pohon biner umumnya diimplementasikan dengan sebuah struktur dinamis yang mirip dengan implementasi linked list[8].

Struktur pohon biasanya diilustrasikan sebagai struktur abstrak memakai simpul-simpul yang diwakili dengan lingkaran atau kotak dan pinggir yang diwakili dengan garis atau panah. Untuk mengimplementasikan sebuah pohon biner, pada tiap simpul kita harus menyimpan isi data simpul itu serta tautan ke kedua anaknya. Dengan demikian kita akan terlebih dahulu mendefinisikan kelas penyimpanan `_SimpulPohonBiner` untuk membuat simpul di sebuah pohon biner dengan kode di bawah ini.

```
1 class _SimpulPohonBiner(object):
2     def __init__( self, data ):
3         self.data = data
4         self.kiri = None
5         self.kanan = None
```

Seperti kelas penyimpan yang lainnya, kelas simpul pohon ini hanya digunakan untuk keperluan internal struktur pohon yang lebih besar. Gambar 9.10 menggambarkan implementasi fisik pohon biner di Gambar 9.4.

[8]Lihat halaman 33 untuk topik ini

### Sebuah contoh membangun pohon biner

Telah kita ketahui bahwa simpul sebuah pohon biner dibangun dari sebuah class yang di dalamnya terdapat data, tautan ke anak kiri, dan tautan ke anak kanan. Untuk membangun dan mengisi sebuah pohon biner, yang kita lakukan adalah membuat simpul-simpulnya (sekaligus mengisi data-nya, kalau kita merujuk pada class di atas), lalu mentautkan satu simpul dengan simpul lain melalui hubungan ortu-anak.

Pada kode di bawah, mula-mula kita membuat simpul-simpul yang `data`-nya adalah beberapa nama kota/kabupaten di Indonesia (tiap simpul berisi *string* nama sebuah kota/kabupaten). Sesudah itu kita menghubungkan simpul-simpul yang baru saja kita buat, dengan cara mengisi `kiri` dan `kanan` pada simpul-simpul yang berperan sebagai ortu.

```
# Membuat simpul-simpul dan mengisi data
A = _SimpulPohonBiner('Ambarawa')
B = _SimpulPohonBiner('Bantul')
C = _SimpulPohonBiner('Cimahi')
D = _SimpulPohonBiner('Denpasar')
E = _SimpulPohonBiner('Enrekang')
F = _SimpulPohonBiner('Flores')
G = _SimpulPohonBiner('Garut')
H = _SimpulPohonBiner('Halmahera Timur')
I = _SimpulPohonBiner('Indramayu')
J = _SimpulPohonBiner('Jakarta')

# Menghubungkan simpul ortu-anak
A.kiri = B; A.kanan = C
B.kiri = D; B.kanan = E
C.kiri = F; C.kanan = G
E.kiri = H
G.kanan = I
```

Verifikasilah bahwa pohon di atas membentuk pohon yang ditunjukkan di Gambar 9.5a.

**Latihan 9.4** Kerjakan yang berikut ini.

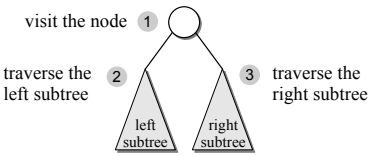
a. Dengan mengacu pada kode di atas, buatlah pohon biner yang diilustrasikan pada Gambar 9.5b dan Gambar 9.5c.

b. Buatlah sebuah pohon biner sempurna dari sebagian simpul-simpul di atas. (Mengapa "sebagian"?)

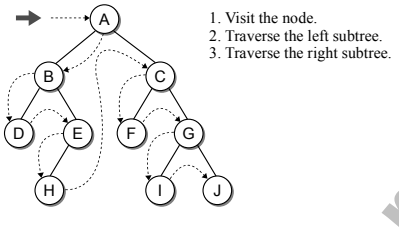
□

## 9.5 *Tree Traversals*

Yang dimaksud *tree traversals* di sini adalah mengakses setiap simpul-simpul di suatu pohon, satu demi satu. Operasi traversal ini adalah salah operasi yang paling sering dilakukan. Operasi yang sesungguhnya dilakukan saat "mengunjungi" tiap simpul adalah bergantung pada aplikasinya. Namun bisa jadi hanya sekedar mencetak data di tiap simpul atau menyimpan data itu ke suatu berkas.




**Gambar** 9.11: Struktur pohon akan di-traverse secara rekursif. Pertama kunjungi simpulnya. Kedua, traverse subpohon kiri. Ketiga, traverse subpohon kanan.




**Gambar** 9.12: Pengurutan simpul-simpul pada *preorder traversal*.

Kalau kita punya struktur linier seperti daftar bertautan (*linked lists*, hlm. 33), proses traversal agak lebih mudah dibayangkan. Kita tinggal memulai dari simpul (elemen) pertama dan mengunjungi satu persatu melalui tautan antar simpul, sampai simpul terakhir. Tapi bagaimanakah kita mengunjungi setiap simpul di sebuah pohon biner? Di situ tidak ada jalur tunggal dari akar menuju simpul lain di pohon itu. Jika kita mengikuti tautannya begitu saja, maka sekali kita sampai di sebuah simpul daun, kita tidak akan bisa secara langsung mengakses simpul-simpul lain di pohon itu.

### 9.5.1 *Preorder Traversal*

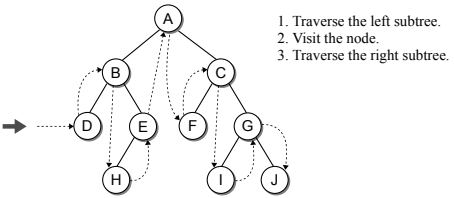
Traversal pohon haruslah dimulai dari simpul akar, karena hanya dari situlah kita bisa mengakses pohonnya secar keseluruhan. Sesudah mengunjungi simpul akar, kita lalu mengunjungi simpul-simpul[9] di subpohon sebelah kiri diikuti subpohon di sebelah kanan. Karena setiap simpul adalah akar subpohon miliknya, kita dapat mengulang proses yang sama pada tiap simpul, yang menghasilkan solusi rekursif[10]. Kejadian dasar (*base case*) muncul saat tautan anak (kiri maupun kanan) yang ditemui mempunyai nilai `None`, karena ini berarti tidak terdapat lagi subpohon yang harus diproses lewat tautan itu. Operasi rekursif ini dapat ditampilkan secara grafis seperti yang ditunjukkan di Gambar 9.11.

Perhatikan pohon biner yang ditunjukkan di Gambar 9.12. Garis putus-putus di situ menunjukkan pengurutan logikal simpul-simpul saat dikunjungi: `A`, `B`, `D`, `E`, `H`, `C`, `F`, `G`, `I`, `J`.

Fungsi rekursif untuk *preorder traversal* sebuah pohon biner adalah cukup sederhana, seperti ditunjukkan pada kode di bawah. Argumen `subpohon` hanya mempunyai dua kemungkinan nilai: yaitu `None`, atau kalau bukan itu, berupa tautan ke akar subpohon berikutnya.

[9]Sekali lagi, yang dimaksud "mengunjungi simpul" adalah melakukan operasi yang diperlukan (hal ini bergantung aplikasinya) di simpul itu. Pada contoh-contoh di sini, operasinya adalah menge-`print` data di simpul itu. Aplikasi lain mungkin akan menyimpan file, mengirim email, menggambar garis, dsb.

[10]Lihat Kotak di halaman 57 untuk paparan sekilas fungsi rekursif.




**Gambar** 9.13: Pengurutan simpul-simpul pada *inorder traversal*.

```
# preorder traversal
def preorderTrav( subpohon ):
    if subpohon is not None :
        print( subpohon.data )
        preorderTrav( subpohon.kiri )
        preorderTrav( subpohon.kanan )
```

Jika isinya bukan `None`, simpul itu dikunjungi terlebih dahulu dan lalu dua subpohonnya dikunjungi. Dari kesepakatan, kita tentukan bahwa yang dikunjungi dulu adalah subpohon kiri sebelum yang kanan. Argumen `subpohon` akan mempunyai nilai `None` saat pohon biner si subpohon itu kosong atau program kita mencoba mengikuti tautan kosong salah satu atau kedua anak simpul.

Jika diberi sebuah pohon biner dengan ukuran $n$, sebuah traversal komplet akan mengunjungi setiap simpul masing-masing satu kali.

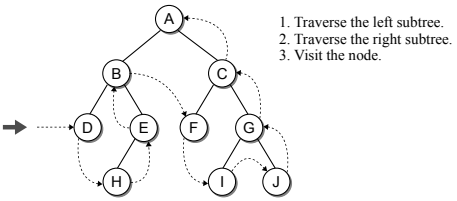
### 9.5.2 *Inorder Traversal*

Pada preorder tranversal di atas kita memilih untuk mengunjungi simpulnya dulu baru men-traverse kedua subpohonnya. Cara lain untuk men-traverse sebuah pohon biner adalah dengan ***inorder traversal*** di mana kita pertama-tama men-traverse subpohon sebelah kiri, lalu mengunjungi simpulnya[11], dilanjutkan men-traverse subpohon sebelah kanan. Gambar 9.13 menunjukkan urutan kunjungan ke simpul-simpul pada pohon biner yang menjadi contoh: `D`, `B`, `H`, `E`, `A`, `F`, `C`, `I`, `G`, `J`.

Fungsi rekursif untuk *inorder traversal* sebuah pohon biner tertuang pada kode di bawah. Kodenya hampir sama dengan kode untuk preorder traversal. Perbedaannya adalah operasi kunjungannya berpindah ke tengah, menjadi sesudah traversal subpohon kiri.

```
# inorder traversal
def inorderTrav( subpohon ):
    if subpohon is not None :
        inorderTrav( subpohon.kiri )
        print( subpohon.data )
        inorderTrav( subpohon.kanan )
```

[11] lihat maksud "mengunjungi simpul" di Catatan Kaki 9 di hlm. 94.




**Gambar** 9.14: Pengurutan simpul-simpul pada *postorder traversal*.

### 9.5.3 *Postorder Traversal*

Kita juga bisa melakukan *postorder traversal*, di mana subpohon kiri dan kanan di tiap simpul ditraverse dulu sebelum simpulnya dikunjungi[12]. Fungsi rekursifnya diperlihatkan di bawah.

```
# postorder traversal
def postorderTrav( subpohon ):
    if subpohon is not None :
        postorderTrav( subpohon.kiri )
        postorderTrav( subpohon.kanan )
        print( subpohon.data )
```

Pohon contoh dengan urutan pengunjungan simpul memakai *postorder traversal* diperlihatkan di Gambar 9.14. Simpul-simpulnya dikunjungi dengan urutan `D`, `H`, `E`, `B`, `F`, `I`, `J`, `G`, `C`, `A`. Perhatikan bahwa pada preorder traversal simpul akarnya selalu dikunjungi pertama sedangkan pada postorder traversal simpul akarnya dikunjungi terakhir.

**Latihan 9.5** Perhatikan kembali kode pada halaman 93 (contoh membangun sebuah pohon biner). Cetaklah semua data di simpul-simpulnya dengan cara *preorder traversal*, *inorder traversal*, dan *postorder traversal*. Verifikasilah bahwa yang tercetak adalah sesuai dengan tiga gambar keterangan yang sudah disampaikan. □

## 9.6 Soal-soal untuk Mahasiswa

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

1. Diberikan pohon biner dengan ukuran $n$, berapakah jumlah level minimum yang bisa dimuatnya? Berapakah jumlah level maksimumnya? Tentukan untuk nilai $n$ berikut.

   (a) $n = 10$ (b) $n = 35$ (c) $n = 76$ (d) $n = 345$

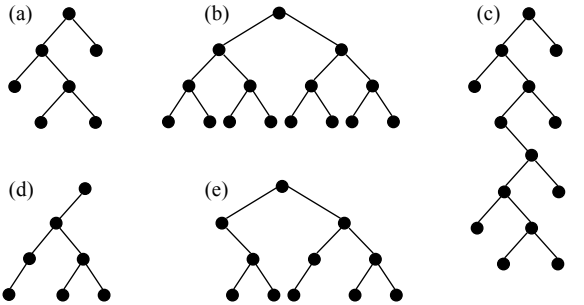
2. Gambarlah semua bentuk pohon biner berukuran 5 yang mungkin. Ada berapa kemungkinan?

3. Berapakah jumlah simpul maksimum suatu pohon biner dengan jumlah level $h$? Tentukan untuk nilai $h$ berikut.

[12] lihat maksud "mengunjungi simpul" di Catatan Kaki 9 di hlm. 94.

(a) $h = 3$ (b) $h = 4$ (c) $h = 5$ (d) $h = 6$

4. Diberikan pohon-pohon biner seperti di bawah.



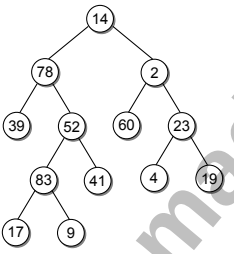

(a) Tunjukkan semua properti struktural yang berlaku pada tiap-tiap pohon di atas: *penuh*, *sempurna*, *komplet*. Ingat bahwa sebuah pohon biner bisa saja bersifat penuh sekaligus sempurna, dan sebagainya.

(b) Tentukan ukuran tiap pohon.

(c) Tentukan ketinggian tiap pohon.

(d) Tentukan lebar tiap pohon.

5. Perhatikan pohon biner berikut.




(a) Tunjukkan urutan pengunjungan simpul untuk

i. preorder traversal ii. inorder traversal iii. postorder traversal

(b) Simpul mana saja yang merupakan simpul daun?

(c) Simpul mana saja yang merupakan simpul dalam?

(d) Simpul mana saja yan berada di level 4?

(e) Tulis semua simpul yang berada di dalam jalur dari simpul akar menuju simpul

i. 83 ii. 39 iii. 4 iv. 9

(f) Perhatikan simpul 52. Tentukan

i. keturunannya ii. leluhurnya iii. saudaranya

(g) Tentukan kedalaman dari tiap-tiap simpul ini:

i. 78 ii. 41 iii. 60 iv. 19

## *Soal-soal pemrograman*

6. Buatlah fungsi `ukuranPohon(akar)` yang akan mendapatkan ukuran sebuah pohon biner.
7. Buatlah sebuah fungsi `tinggiPohon(akar)` yang akan mendapatkan ketinggian sebuah pohon biner.
8. Buatlah sebuah fungsi yang mencetak data tiap simpul sekaligus level di mana simpul itu berada. Silakan memilih akan memakai *preorder traversal*, *inorder traversal*, atau *postorder traversal*. Contoh sepotong hasilnya adalah seperti di bawah ini (jika kamu memakai *preorder traversal*).

```
>>> cetakDataDanLevel(A)
Ambarawa, level 0
Bantul, level  1
Denpasar, level 2
Enrekang, level 2
Halmahera Timur, level 3
Cimahi, level 1
```

# Modul 10

# Analisis Algoritma

Untuk setiap soal dan tantangan dalam pemrograman dan ilmu komputer, terdapat banyak algoritma yang bisa menyelesaikan soal dan tantangan itu. Pertanyaannya lalu: algoritma mana yang lebih cepat dan efisien? Algoritma mana yang lebih stabil? Algoritma mana yang lebih cocok untuk keadaan tertentu?

Analisis algoritma akan mempelajari pembandingan algoritma-algoritma berdasarkan banyaknya sumberdaya (memori, waktu, ...) yang dipakai tiap-tiap algoritma. Waktu yang diperlukan untuk menjalankan suatu algoritma seringkali bergantung pada banyaknya data yang diolah. Banyaknya data ini disimbolkan dengan $n$. Untuk suatu permasalahan, ada algoritma yang waktu pengerjaannya sebanding dengan $n$ (saat $n$ naik 10 kali, waktunya naik 10 kali). Untuk permasalahan yang lain, ada algoritma yang sebanding dengan $n^2$ (saat $n$ naik 10 kali, waktunya naik 100 kali). Ada pula yang sebanding dengan $n \log n$. Ada yang tidak bergantung pada $n$. Kita ingin algoritma yang efisien, yakni menjalankan tugasnya dengan benar dan secepat mungkin.

## 10.1 Sebuah contoh

### Menjumlahkan bilangan 1 sampai $n$

Kamu diminta membuat program –memakai Python– yang akan menjumlahkan semua bilangan bulat dari 1 sampai $n$. Misal $n = 10$, maka si program akan mengembalikan 55 (karena $55 = 1 + 2 + \cdots + 10$). Bagaimana kamu membuatnya?

Cara pertama adalah cara 'lugu', yakni dengan menjumlahkannya betul-betul, seperti di bawah ini. (Ketiklah.)

```
1 def jumlahkan_cara_1(n):
2     hasilnya = 0
3     for i in range(1, n+1):
4         hasilnya = hasilnya + i
5     return hasilnya
```

Ketika dijalankan, tentu dia akan mengembalikan hasil yang diharapkan.

```
>>> jumlahkan_cara_1(10)
55
>>> jumlahkan_cara_1(100)
5050
```

Bagaimanakah kinerja (performance) algoritma ini? Jika $n$ semakin besar, apakah perlu waktu yang lebih lama? Mestinya iya. Mari kita coba untuk mengukur waktu eksekusinya.

Pengukuran kinerja bisa dilakukan dengan berbagai cara. Salah satunya adalah dengan menandai waktu di awal ekeskusi dan di akhir ekesekusi, lalu membandingkan keduanya. Secara praktis, kita meng-import module `time` dan "membungkus" perintah eksekusi dengan penanda waktu. Tambah kode di atas untuk menjadi seperti di bawah ini.

```
1  import time
2
3  def jumlahkan_cara_1(n):
4      hasilnya = 0
5      for i in range(1, n+1):
6          hasilnya = hasilnya + i
7      return hasilnya
8
9  for i in range(5):                # mengulang lima kali
10     awal = time.time()            #     menandai awal kerja
11     h = jumlahkan_cara_1(10000)   #     menjumlah 1 sampai sepuluh ribu
12     akhir = time.time()           #     menandai akhir kerja, lalu mencetak
13     print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, akhir-awal))
```

Hasilnya adalah seperti di bawah ini (akan berbeda di komputermu):

```
Jumlah adalah 50005000, memerlukan 0.00400043 detik
Jumlah adalah 50005000, memerlukan 0.00200009 detik
Jumlah adalah 50005000, memerlukan 0.00199986 detik
Jumlah adalah 50005000, memerlukan 0.00099993 detik
Jumlah adalah 50005000, memerlukan 0.00200033 detik
>>>
```

Coba naikkan $n$ menjadi satu juta (baris 11), lalu jalankan:

```
Jumlah adalah 500000500000, memerlukan 0.16300941 detik
Jumlah adalah 500000500000, memerlukan 0.15600896 detik
Jumlah adalah 500000500000, memerlukan 0.15900898 detik
Jumlah adalah 500000500000, memerlukan 0.15800905 detik
Jumlah adalah 500000500000, memerlukan 0.15700889 detik
>>>
```

Seperti diduga, perlu waktu lebih lama untuk menjumlahnya. Itu adalah cara pertama.

Sekarang kita akan mengetik algoritma yang lebih baik. Ingat bahwa jumlahan dari 1 sampai $n$ bisa diringkas dengan rumus

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

sehingga kita mempunyai `jumlahkan_cara_2` sebagai berikut

```
1 def jumlahkan_cara_2(n):
2     return ( n*(n + 1) )/2
```

yang kalau kita tes dengan cara yang sama, kita akan mendapatkan seperti yang di bawah

```
Jumlah adalah 500000500000, memerlukan 0.00000001 detik
Jumlah adalah 500000500000, memerlukan 0.00000000 detik
Jumlah adalah 500000500000, memerlukan 0.00000000 detik
Jumlah adalah 500000500000, memerlukan 0.00000000 detik
Jumlah adalah 500000500000, memerlukan 0.00000000 detik
>>>
```

Cobalah mengubah $n$ menjadi seratus juta atau satu milyar. Apa yang terjadi? Waktu pengerjaannya tidak akan berubah. Algoritma di cara 2 ini tidak terpengaruh besarnya $n$.

Namun apa arti *benchmark* ini? Secara intuitif bisa kita lihat bahwa penyelesaian yang iteratif (cara 1) bekerja lebih banyak karena ada langkah program yang diulang-ulang. Juga, waktu yang diperlukan untuk menghitung di cara 1 bertambah dengan naiknya $n$.

Apakah angka kecepatan eksekusi di atas akan selalu di sekitar itu? Tidak. Jika kita menjalankan algoritma di atas di komputer yang berbeda, atau menerapkannya dengan bahasa pemrograman yang berbeda, kemungkinan kita akan mendapatkan hasil waktu yang berbeda. Kalau komputer yang dipakai lebih tua, waktu eksekusinya bisajadi lebih lama.

Dengan demikian kita memerlukan sebuah alat ukur yang lebih baik untuk meng-karakterisasi algoritma-algoritma ini, dikaitkan dengan kecepatan eksekusinya. Karakterisasi ini terlepas dari bahasa pemrograman atau komputer yang dipakai. Pengukuran ini dengan demikian berguna untuk membandingkan antar algoritma yang implementasinya bisa di mana saja.

**Pelajaran terpetik**

- Dari contoh di atas, kita bisa melihat bahwa untuk suatu masalah, bisajadi ada dua atau lebih algoritma bisa menyelesaikan. *Namun di antaranya ada yang lebih baik.*
- Kita ingin algoritma yang kecepatan pertumbuhannya (*growth rate* -nya) lambat. Artinya, dengan membesarnya $n$ (yakni ukuran input), waktu eksekusinya tidak terpengaruh banyak.
- Kita memerlukan suatu konsep –berikut notasinya– untuk meng-karakterisasi algoritma-algoritma, sehingga kita bisa membandingkan antar algoritma dengan konsisten.

## 10.2 Notasi Big-O

Untuk mulai mengukur secara kuantitatif kinerja suatu algoritma, kita bisa saja mulai dari menghitung banyaknya operasi atau langkah yang dijalani, sebagai fungsi dari $n$. Sebut saja fungsi yang meng-aproksimasi ini adalah $T(n)$. Parameter $n$ sering disebut "besarnya problem" (*size of the problem*).

### Ilustrasi $\mathcal{O}(f(n))$

Untuk algoritma cara 1 di atas, bisa kamu lihat bahwa $T(n) = 1 + n$. Bisa kita baca sebagai "$T(n)$ adalah waktu yang diperlukan untuk menyelesaikan problem ini, dalam hal ini sebanyak $n + 1$". Dengan bertambah besarnya $n$, maka angka 1 nya menjadi semakin kurang signifikan. Maka kita katakan bahwa algoritma ini, dalam notasi big-O, mempunyai *running time* $\mathcal{O}(n)$.

Sebuah contoh lain. Misal suatu algoritma mempunyai banyak operasi[1] yang dirumuskan dengan $T(n) = 5n^2 + 32n + 1440$. Saat $n$ kecil, misal 1 atau 2 atau 3, konstanta 1440 tampak dominan. Tapi dengan membesarnya $n$, maka suku $n^2$ menjadi paling dominan. Saat $n$ menjadi besar sekali, dua suku yang lain menjadi tidak signifikan untuk menentukan hasil akhir. Sehingga untuk men-aproksimasi $T(n)$ saat $n$ cukup besar, dua suku lain itu bisa diabaikan dan kita fokus pada $5n^2$. Terlebih lagi, bila kita membandingkan kenaikan $T(n)$ saat $n$ berubah dari nilai besar ke nilai yang lebih besar lagi, maka koefisian angka 5 ini menjadi bisa diabaikan dan *running time* algoritma ini adalah $\mathcal{O}(n^2)$.

Kompleksitas waktu yang sering muncul dalam analisis algoritma adalah $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, $\mathcal{O}(\sqrt{n})$, $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^d)$, dan $\mathcal{O}(2^n)$. Kita akan kembali ke sini lagi nanti.

BTW, algoritma cara 2 di atas mempunyai kompleksitas $\mathcal{O}(1)$.

### Formulasi

Notasi big-O bermakna *"order of magnitude"* dan ditulis dalam bentuk $\mathcal{O}(f(n))$. Idenya adalah kita ingin dapat mengungkapkan bagaimana waktu eksekusi suatu algoritma tumbuh semakin besar dengan naiknya $n$. Misal di satu contoh di atas, kita bisa mengatakan "algoritma ini *running time*-nya tumbuh sejalan dengan $n$ kuadrat." Atau kita bisa mengungkapkan "Algoritma pertama *tumbuhnya* lebih cepat daripada algoritma kedua –artinya algoritma kedua lebih baik."

Awalnya adalah mencari sebuah fungsi yang dapat "mengatasi" atau "mengalahkan" $T(n)$ saat $n$ membesar, dan sekaligus sedekat mungkin terhadap $T(n)$.

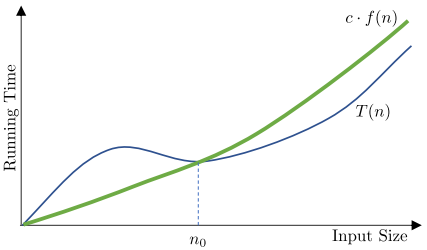
**Definisi 10.1** Misalkan $T(n)$ dan $f(n)$ sebagai fungsi yang memetakan dari bilangan integer positif ke bilangan real. Maka kita katakan bahwa $T(n)$ mempunyai order $\mathcal{O}(f(n))$ jika ada konstanta real $c$ dan konstanta integer $n_0 \geq 1$ sedemikian rupa sehingga

$$T(n) \leq c \cdot f(n), \quad \text{untuk} \quad n \geq n_0.$$

Lihat Gambar 10.1. □

Sebagai tambahan, kita akan mencari suatu fungsi $f(n)$ yang *paling simple* yang lalu $c \cdot f(n)$ bisa menjadi *batas atas* yang paling ketat bagi fungsi $T(n)$.

[1] Untuk saat ini, tidak penting benar jenis operasinya; penjumlahan atau perkalian atau apapun. Yang penting bagaimana waktu eksekusinya *tumbuh* dengan semakin besarnya ukuran input.




**Gambar** 10.1: Ilustrasi notasi Big-O. Fungsi $T(n)$ berada dalam $\mathcal{O}(f(n))$, karena $T(n) \leq c \cdot f(n)$ untuk semua $n \geq n_0$. Dengan kata lain: untuk semua $n$ di sebelah kanan $n_0$, nilai $c \cdot f(n)$ selalu 'mengalahkan' $T(n)$. Sebagai tambahan, $c \cdot f(n)$ haruslah seketat mungkin ke $T(n)$, dan fungsi $f(n)$ haruslah se-simple mungkin.

**Contoh 10.2** Fungsi $T(n) = 25n^4 + 8n^3 + 7n^2 + 9n + 4$ mempunyai order $\mathcal{O}(n^4)$.

Perhatikan bahwa $25n^4 + 8n^3 + 7n^2 + 9n + 4 \leq (25 + 8 + 7 + 9 + 4)n^4 = cn^4$ untuk $c = 53$ dan $n_0 = 1$. □

Sesungguhnya, kita bisa mengkarakterisasi kecepatan pertumbuhan semua fungsi polinomial.

**Proposisi 10.3** Jika $T(n)$ adalah polinomial dengan derajat $d$, yakni,

$$T(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0,$$

di mana $a_d > 0$, maka $T(n)$ berada dalam $\mathcal{O}(n^d)$.

**Sketsa bukti.** Perhatikan bahwa untuk $n \geq 1$, kita mempunyai $n^d \geq \cdots \geq n^2 \geq n \geq 1$. Sehingga

$$a_d n^d + a_{d-1} n^{d-1} + \cdots + a_2 n^2 + a_1 n + a_0 \leq \big( |a_d| + |a_{d-1}| + \cdots + |a_2| + |a_1| + |a_0| \big) n^d.$$

Lalu kita tunjukkan bahwa $f(n)$ berada dalam $\mathcal{O}(n^d)$ dengan memilih konstanta $c = |a_d| + \cdots + |a_1| + |a_0|$ dan $n_0 = 1$. □

Jadi, pangkat tertinggilah yang menentukan kecepatan pertumbuhan polinomial itu. Di bawah ini ada beberapa contoh lagi, yang melibatkan fungsi fundamental lain. Kita bersandar pada fakta bahwa $\log n \leq n$ untuk $n \geq 1$. Di sini, fungsi log mempunyai basis 2. Jadi, sebagai contoh, $\log 32 = 5$ karena $2^5 = 32$.

**Contoh 10.4** $3n^2 + 5n \log n + 6n + 7$ berada dalam $\mathcal{O}(n^2)$. □

**Contoh 10.5** $12n^3 + 10n \log n + 9$ berada dalam $\mathcal{O}(n^3)$. □

**Contoh 10.6** $5 \log n + 3$ berada dalam $\mathcal{O}(\log n)$.

Perhatikan bahwa $8 \log n \leq 5 \log n + 3$ untuk $n \geq 2$. Ingat bahwa $\log n$ bernilai 0 saat $n = 1$. Itulah mengapa kita menggunakan $n \geq n_0 = 2$ di sini. □

**Tabel** 10.1: Beberapa fungsi umum $\mathcal{O}(f(n))$, diurutkan dari waktu eksekusi cepat (kompleksitas rendah) ke waktu eksekusi semakin lambat (kompleksitas tinggi). Di sini log bermakna $\log_2$, yakni logaritma berbasis 2.

| $f(n)$ | Nama umum | Contoh algoritma |
|---|---|---|
| 1 | constant | Append, get item, set item. Menentukan suatu bilangan genap atau ganjil. Menemukan nilai median di suatu list yang sudah urut. Menghitung $(-1)^n$. |
| $\log n$ | logarithmic | Menemukan suatu item di sebuah list yang sudah urut menggunakan *binary search*. |
| $n$ | linear | *Copy, insert, delete, iteration.* Menemukan sebuah item di sebuah list yang tidak urut. |
| $n \log n$ | log-linear | Merge sort dan Quick sort (kasus rata-rata). *Fast Fourier Transform.* |
| $n^2$ | quadratic | Mengalikan dua bilangan $n$ digit dengan algoritma sederhana. Selection sort. Insertion sort. Batas (di kasus terburuk) untuk algoritma yang biasanya cepat seperti Quick sort. |
| $n^3$ | cubic | *Tree-adjoining grammar parsing.* Perkalian dua matriks $n \times n$ dengan 'cara lugu'. |
| $2^n$ | exponential | Solusi exact untuk *travelling salesman problem* memakai *dynamic programming. Towers of Hanoi problem.* |

**Contoh 10.7** $2^{n+2}$ berada dalam $\mathcal{O}(2^n)$.

Perhatikan bahwa $2^{n+2} = 2^2 \cdot 2^n = 4 \cdot 2^n$. □

**Contoh 10.8** $2n + 35 \log n$ berada dalam $\mathcal{O}(n)$. □

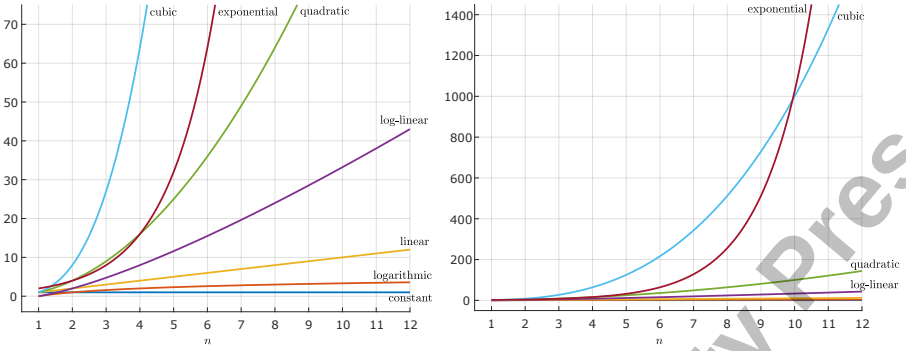
### Fungsi yang paling simple dan juga paling ketat

Umumnya kita harus menemukan notasi $\mathcal{O}(\cdot)$ yang sedekat mungkin dengan fungsi yang akan dijelaskan. Sebagai contoh, jika kita punya $T(n) = 5n^3 + 3n^2 + 4n + 5$, maka bisa saja kita katakan bahwa $T(n)$ berada dalam $\mathcal{O}(n^5)$ atau berada dalam $\mathcal{O}(n^4)$. Tapi akan lebih tepat kalau kita katakan $T(n)$ berada dalam $\mathcal{O}(n^3)$. Analoginya seperti di bawah.

Ada yang bertanya, "berapa lamakah berjalan dari Gedung J Lantai 1 ke Masjid Sudalmiyah Rais?" Ini bisa dijawab: "kurang dari sehari," atau "tidak lebih dari 1 jam." Ini secara teknis benar semua. Tapi akan lebih tepat (dan lebih membantu) kalau dijawab "berjalanlah ke arah utara dan kamu akan sampai dalam waktu kurang dari dua menit."

### Perbandingan Fungsi-fungsi $\mathcal{O}(f(n))$ yang Umum

Terdapat beberapa fungsi yang umum ditulis dalam analisis algoritma. Ini ditampilkan dalam Tabel 10.1 dan grafik pertumbuhannya ditampilkan pada Gambar 10.2.




**Gambar** 10.2: Pertambahan kompleksitas / waktu ekeskusi dengan membesarnya $n$. Kiri: *zooom-in* sumbu $y$ grafik. Kanan: *zoom-out* sumbu $y$ grafik. Tampak bahwa exponential pada suatu saat akan 'mengatasi' cubic. Lebih umum lagi –ini suatu fakta– adalah bahwa fungsi *exponential* akan 'mengatasi' fungsi *polynomial* $n^d$ apapun, jika $n$ cukup besar. Jadi, suatu kabar buruk jika suatu algoritma mempunyai kompleksitas *exponential* $\mathcal{O}(2^n)$.

## 10.3 Kasus terburuk, rata-rata, dan terbaik

Kinerja suatu algortima seringkali bergantung pada nilai sebenarnya di datanya, tidak semata-mata pada ukuran/banyaknya data. Untuk algoritma yang seperti ini, kita perlu membedakan kinerjanya saat menghadapi kasus terbaik (*best case*), kasus terburuk (*worst case*), atau kasus rata-rata (*average case*). Ada algoritma yang pada kasus terburuk mempunyai kinerja $\mathcal{O}(n^2)$, namun pada kasus rata-rata kinerjanya adalah $\mathcal{O}(n \log n)$.

Sebagai contoh, untuk sebuah algoritma pengurutan[2],

- Kasus terbaik adalah saat datanya sudah urut atau hampir urut.
- Kasus terburuk adalah saat data yang diterima terbalik urutannya.
- Kasus rata-rata adalah saat datanya acak.

Mari kita lihat perbedaan ini pada algoritma *Insertion Sort* yang sudah dibahas di Bab 5. Kita mulai dari *average case scenario* dulu, di mana data yang masuk berposisi acak. Siapkan kode *Insertion Sort* yang sudah kamu buat (dengan meng-*import* atau mengkopi kodenya), lalu ketik yang di bawah ini.

```
for i in range(5):
    L = list(range(3000))
    random.shuffle(L)      # Mengacak posisi elemen di list
    awal = time.time()
    U = insertionSort(L)
    akhir = time.time()
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L),akhir-awal))
```

Yang akan menghasilkan

[2]Tidak semua algoritma pengurutan mempunyai sifat seperti ini. Ada yang tidak terpengaruh urutan awal. Ada yang jika di awal urutannya terbalik, maka ini justru termasuk salah satu kasus terbaik.

```
Mengurutkan 3000 bilangan, memerlukan 1.1620665 detik
Mengurutkan 3000 bilangan, memerlukan 1.1730671 detik
Mengurutkan 3000 bilangan, memerlukan 1.1550660 detik
Mengurutkan 3000 bilangan, memerlukan 1.1370649 detik
Mengurutkan 3000 bilangan, memerlukan 1.1420655 detik
>>>
```

Sesudah itu, kita coba yang *worst case scenario* untuk algoritma ini. Yakni data yang masuk urutannya terbalik. Ikuti yang berikut.

```
for i in range(5):
    L = list(range(3000))
    L = L[::-1]         # Membalik urutan elemen di list
    awal = time.time()
    U = insertionSort(L)
    akhir = time.time()
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L),akhir-awal))
```

yang menghasilkan

```
Mengurutkan 3000 bilangan, memerlukan 2.2251272 detik
Mengurutkan 3000 bilangan, memerlukan 2.2221272 detik
Mengurutkan 3000 bilangan, memerlukan 2.2151270 detik
Mengurutkan 3000 bilangan, memerlukan 2.2291274 detik
Mengurutkan 3000 bilangan, memerlukan 2.2141266 detik
>>>
```

Sekarang kita coba *best case scenario*, di mana data masukannya sudah sejak awal urut. Ikuti yang berikut ini.

```
for i in range(5):
    L = list(range(3000))

    awal = time.time()
    U = insertionSort(L)
    akhir = time.time()
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L),akhir-awal))
```

yang menghasilkan

```
Mengurutkan 3000 bilangan, memerlukan 0.0020003 detik
Mengurutkan 3000 bilangan, memerlukan 0.0009999 detik
Mengurutkan 3000 bilangan, memerlukan 0.0020001 detik
Mengurutkan 3000 bilangan, memerlukan 0.0020003 detik
Mengurutkan 3000 bilangan, memerlukan 0.0020001 detik
>>>
```

Terlihat bahwa *insertion sort* mempunyai kinerja yang berbeda-beda antara *average case, worst case*, dan *best case*.

Tidak semua algoritma pengurutan mempunyai sifat seperti ini. Fungsi pengurutan yang ada di Python, `sorted()`, yang memakai algoritma Timsort[3], mempunyai sifat yang menarik. Lihat bagian latihan di akhir modul.

[3] https://en.wikipedia.org/wiki/Timsort

## 10.4 Menganalisis kode Python

Seperti sudah diterangkan sebelumnya, analisis algoritma bermula dari operasi-operasi dasar (*basic operations*). Tapi apa yang dimaksud dengan operasi dasar? Operasi dasar adalah pernyataan dan pemanggilan fungsi yang waktu eksekusinya tidak bergantung pada nilai spesifik data yang sedang dimanipulasi. Berikut ini adalah contoh operasi dasar, tiap-tiap barisnya.

```
x = 5
y = x
z = x + y*8
d = x > 0 and x < 100
f = [3,2,4,5]
v = f[0:2]
```

Ketika ada dua operasi yang berurutan, maka kompleksitasnya dijumlahkan. Sebagai contoh, perhatikan dua operasi berurutan: penyisipan suatu item ke sebuah list dan lalu mengurutkan list itu. Kita tahu penyisipan satu item memakan waktu $\mathcal{O}(n)$ dan pengurutan memerlukan waktu $\mathcal{O}(n \log n)$. Total kompleksitas waktunya adalah menjadi $\mathcal{O}(n + \log n)$. Namun kita hanya memperhatikan fungsi yang order-nya lebih tinggi[4], sehingga kompleksitasnya kita tulis menjadi $\mathcal{O}(n \log n)$.

Jika kita mengulang suatu operasi, misal di `while`-loop atau `for`-loop, maka kita mengalikan kelas kompleksitasnya dengan banyaknya operasi.

Sebagai contoh, misal fungsi `myFun()` mempunyai kompleksitas waktu $\mathcal{O}(n^2)$, dan dieksekusi sebanyak $n$ kali di dalam sebuah for-loop:

```
for i in range(n):
    myFun(...)
```

maka kompleksitas waktunya adalah $\mathcal{O}(n \cdot n^2) = \mathcal{O}(n^3)$.

Jika kita mempunyai loop di dalam loop (*nested loop*), maka kodenya akan dijalankan $n^2$ kali, dengan anggapan kedua loop berjalan $n$ kali. Sebagai contoh:

```
for i in range(n):
    for j in range(n):
        # pernyataan
```

Jika setiap pernyataan adalah konstan, dan dijalankan $n \cdot n$ kali, maka bisa kita tulis bahwa kompleksitasnya adalah $\mathcal{O}(n^2)$.

Kode berikut mempunyai kompleksitas $\mathcal{O}(\log n)$

```
count = 0
i = 32
while i >= 1:
    count = count + 1
    i = i//2
print(count)
```

Perhatikan bahwa setiap kali menghitung, nilai $i$ dibagi 2. Ini akan membuat programnya lebih cepat selesai dibandingkan jika $i$ dikurangi satu tiap penghitungan. Cobalah eksekusi kode di

[4] Yakni, fungsi yang 'mengatasi' saat $n$ besar.

atas dengan $n = 100$. Di sini ukuran input dikecilkan jadi separuhnya setiap kali dijalankan ulang, sehingga untuk mencapai ukuran 1 banyaknya iterasi yang dibutuhkan adalah

$$\lfloor \log_2 n \rfloor + 1$$

yakni bilangan bulat terbesar yang kurang dari atau sama dengan[5] $\log_2 n$, tambah 1. Jika $n = 100$ maka akan ada iterasi sebanyak $\lfloor \log_2 100 \rfloor + 1 = 6 + 1 = 7$.

## 10.5 Analisis Pewaktuan Menggunakan `timeit`

Module `timeit` dapat membantu untuk secara praktis melakukan pengukuran waktu eksekusi untuk kode-kode sederhana. Perhatikan dan ketiklah kode berikut, serta pahami cara kerjanya.

```
>>> from timeit import timeit
>>> timeit('sqrt(2)' , 'from math import sqrt' , number=10000)
0.00342316301248502
>>> timeit('sqrt(2)' , 'from math import sqrt' , number=100000)
0.019461828997009434
>>> timeit('sqrt(2)' , 'from math import sqrt' , number=1000000)
0.1989576570049394
```

Pada contoh di atas, argumen pertama pada timeit adalah kode yang dijalankan / yang sedang dites. Argumen kedua, `setup`, adalah persiapan sebelum kode yang dites dijalankan (persiapan ini hanya sekali). Argumen kata kunci `number` adalah banyaknya pengulangan, default-nya satu juta jika argumen ini tidak diikutkan. Fungsi ini mengembalikan waktu dalam detik. Lanjutkan eksplorasi kamu dengan `timeit` ini.

```
>>> timeit("1+2")   # Waktu untuk menghitung 1+2, diulang 1 juta kali.
0.01901585698942654
>>> timeit("sin(pi/3)", setup="from math import sin, pi")# sin(pi/3) diulang 1 juta kali
0.2774660010036314
>>> timeit("sin(1.047)", setup="from math import sin")# sin(1.047) diulang 1 juta kali
0.1839345560001675
```

Yang `sin(1.047)` lebih cepat dari `sin(pi/3)` karena tidak perlu operasi pembagian.

### 10.5.1 Melihat $\mathcal{O}(n^2)$ pada *nested loop*

Pada bagian ini kita akan menggambar hasil uji pewaktuan sebuah fungsi dengan kompleksitas $\mathcal{O}(n^2)$. Pastikan modul `matplotlib` terinstall di komputermu. Jika belum ada, install dari internet: `pip install matplotlib` . Gagasan dasarnya adalah

- Ada fungsi yang akan kita uji. Fungsi ini menerima satu argumen, dan mempunyai kalang bersusuh (*nested loop*).
- Kita buat sebuah kalang lain, yang nilai *i*terasi-nya berjalan dari 1 sampai 1000, yang dipakai untuk menguji fungsi tersebut di atas. Nilai *i*terasi ini akan diumpankan ke fungsi

[5] Lambang $\lfloor x \rfloor$, perintah `math.floor` di Python, bermakna "$x$, jika bukan bilangan bulat, akan dibulatkan ke bilangan bulat di bawahnya" yakni *menuju lantai*. Jika ingin menuju bilangan bulat di atasnya, orang menggunakan $\lceil x \rceil$, perintah `math.ceil` di Python: *menuju plafon*.

yang sedang diuji.

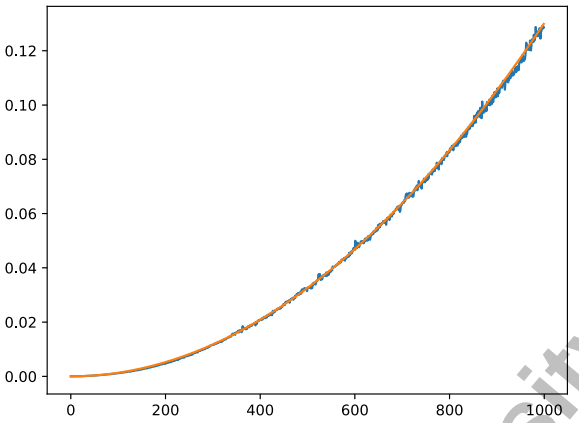
- Untuk tiap $i$terasi, eksekusinya hanya satu kali. Jadi, `number = 1` di `timeit`. Dan lamanya waktu eksekusi dicatat untuk tiap $i$, menggunakan `timeit`.
- Setelah pengujian berakhir, buat grafik yang menggambarkan lamanya waktu eksekusi dari data kecil ke besar.
- Tumpukkan grafik $x^2$ –yang sudah diskala– sebagai pembanding.

Kodenya ada di bawah ini, dan hasilnya ditunjukkan di Gambar 10.3. Ketik dan jalankan.

```
import timeit
import matplotlib.pyplot as plt

## Ini fungsi nested loop yang akan diuji:
def kalangBersusuh(n):
    for i in range(n):
        for j in range(n):
            i+j

## Ini fungsi pengujinya:
def ujiKalangBersusuh(n):
    ls=[]
    jangkauan = range(1,n+1)
    siap = "from __main__ import kalangBersusuh"
    for i in jangkauan:
        print('i =',i)  # baris ini bisa dihidupkan atau dimatikan
        t=timeit.timeit("kalangBersusuh(" + str(i) +")", setup=siap, number=1)
        ls.append(t)
    return ls

## Pemanggilan pengujian:
n  = 1000
LS = ujiKalangBersusuh(n) # dari 1 sampai 1000.
## LS adalah list hasil uji kecepatan, dari n sedikit ke banyak.

## Menggambar grafik. Di bawah ini saja yang diulang saat me-nyetel skala.
plt.plot(LS)    # Mem-plot hasil uji
skala = 7700000  # <------ Setel skala ini sesuai hasilmu.
plt.plot([x*x/skala for x in range(1,n+1)]) # Grafik x^2 untuk pembanding.
plt.show()      # Tunjukkan plotnya
```

Hasil grafik sesuai dengan ekspektasi kita. Perlu diingat bahwa ini mewakili kinerja algoritmanya sekaligus perilaku platform perangkat keras dan lunaknya.

Tentu saja, processor yang lebih cepat akan menghasilkan waktu eksekusi yang lebih cepat. Kinerja juga akan dipengaruhi oleh proses lain yang sedang berjalan, oleh kekangan memori, oleh kecepatan clock, dan lain sebagainya. Namun kenaikan waktu eksekusi relatif dari $n$ yang kecil ke $n$ yang lebih besar akan menunjukkan trend yang kurang lebih seragam. Pemilihan algoritma tidak hanya melihat $\mathcal{O}(\cdot)$, tapi juga kecepatan sebenarnya untuk data yang *typical* akan dipakai saat *production*.




**Gambar** 10.3: Hasil ujicoba kalang bersusuh, yang mempunyai kompleksitas $\mathcal{O}(n^2)$. Hasil akan berbeda angkanya untuk komputer yang berbeda, namun bentuk grafik akan kurang lebih sama.

## 10.6 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, pastikan bahwa kamu paham semua contoh soal di modul ini.

1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul `timeit`, yakni (a) `jumlahkan_cara_1` (b) `jumlahkan_cara_2` (c) `insertionSort`

   Untuk `insertionSort`, kerjakan untuk ketiga kasusnya.

2. Python mempunyai perintah untuk mengurutkan suatu list yang memanfaatkan algoritma Timsort. Jika `g` adalah suatu list berisi bilangan, maka `g.sort()` kan mengurutkannya. Perintah yang lain, `sorted()` mengurutkan list dan *mengembalikan* sebuah list baru yang sudah urut. Selidikilah fungsi `sorted` ini menggunakan `timeit`:

   - Apakah yang merupakan *best case* dan *average case* bagi `sorted()`?
   - Confirm bahwa data input urutan terbalik *bukan* kasus terburuk bagi `sorted()`. Bahkan dia lebih cepat dalam mengurutkannya daripada data input random.

3. Untuk tiap kode berikut, tentukan *running time*-nya, $\mathcal{O}(1)$, $\mathcal{O}(\log n)$, $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$, $\mathcal{O}(n^2)$, atau $\mathcal{O}(n^3)$, atau yang lain. Untuk memulai analisis, ambil suatu nilai $n$ tertentu, lalu ikuti apa yang terjadi di kode itu.

   (a) loop di dalam loop, keduanya sebanyak $n$:

   ```
   test = 0
   for i in range(n):
       for j in range(n):
           test = test + i * j
   ```

   (b) loop di dalam loop, yang dalam bergantung nilai $i$ loop luar:

   ```
   test = 0
   ```

```
for i in range(n):
    for j in range(i):
        test = test + i * j
```

(c) dua loop terpisah:

```
test = 0
for i in range(n):
    test = test + 1
for j in range(n):
    test = test - 1
```

(d) *while* loop yang dipangkas separuh tiap putaran:

```
i = n
while i > 0:
    k = 2 + 2
    i = i // 2
```

(e) loop in a loop in a loop, ketiganya sebanyak $n$:

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            m = i + j + k + 2019
```

(f) loop in a loop in a loop, dengan loop dalam sebanyak nilai loop luar terdekat:

```
for i in range(n):
    for j in range(i):
        for k in range(j):
            m = i + j + k + 2019
```

(g) fungsi ini:

```
for i in range( n ) :
    if i % 3 == 0 :
        for j in range( n / 2 ) :
            sum += j
    elif i % 2 == 0 :
        for j in range( 5 ) :
            sum += j
    else :
        for j in range( n ) :
            sum += j
```

4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat:

$$n \log_2 n \quad 4^n \quad 10 \log_2 n \quad 5n^2 \quad \log_4 n \quad 12n^6 \quad 2^{\log_2 n} \quad n^3$$

5. Tentukan $\mathcal{O}(\cdot)$ dari fungsi-fungsi berikut, yang mewakili banyaknya langkah yang diperlukan untuk beberapa algoritma.

(a) $T(n) = n^2 + 32n + 8$

(b) $T(n) = 87n + 8n$

(c) $T(n) = 4n + 5n \log n + 102$

(d) $T(n) = \log n + 3n^2 + 88$

(e) $T(n) = 3(2^n) + n^2 + 647$

(f) $T(n, k) = kn + \log k$

(g) $T(n, k) = 8n + k \log n + 800$

(h) $T(n, k) = 100kn + n$

6. (Literature Review) Carilah di Internet, kompleksitas metode-metode pada object `list` di Python. *Hint:*
   - Google python list methods complexity. Lihat juga bagian "Images"-nya
   - Kunjungi https://wiki.python.org/moin/TimeComplexity
7. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode `append()` adalah $\mathcal{O}(1)$. Gunakan `timeit` dan `matplotlib`, seperti sebelumnya.
8. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode `insert()` adalah $\mathcal{O}(n)$. Gunakan `timeit` dan `matplotlib`, seperti sebelumnya.
9. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah-suatu-nilai-berada-di-suatu-list mempunyai kompleksitas $\mathcal{O}(n)$. Gunakan `timeit` dan `matplotlib`, seperti sebelumnya.
10. (Literature Review) Carilah di Internet, kompleksitas metode-metode pada object `dict` di Python.
11. (Literature Review) Selain notasi big-O $\mathcal{O}(\cdot)$, ada pula notasi big-Theta $\mathbf{\Theta}(\cdot)$ dan notasi big-Omega $\mathbf{\Omega}(\cdot)$. Apakah beda di antara ketiganya?
12. (Literature Review) Apa yang dimaksud dengan *amortized analysis* dalam analisis algoritma?

*Soal-soal mengambil inspirasi dari buku-buku yang tercantum di Daftar Bacaan.*

# Daftar Bacaan


[1] Benjamin Baka. *Python Data Structures and Algorithms: Improve the performance and speed of your applications*. Packt Publishing, 2017.

[2] Wesley J. Chun. *Core Python Programming*. Prentice Hall, New Jersey, $2^{nd}$ edition, 2006.

[3] Wesley J. Chun. *Core Python Applications Programming*. Prentice Hall, New Jersey, $3^{rd}$ edition, 2012.

[4] Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. *Data Structures and Algorithms in Python*. John Wiley & Sons, New Jersey, 2013.

[5] Narasimha Karumanchi. *Data Structures and Algorithmic Thinking in Python*. CareerMonk Publications, Bombay, 2016.

[6] Kenneth A. Lambert and Martin Osborne. *Fundamentals of Python: From First Programs Through Data Structures*. John Wiley & Sons, New Jersey, 2011.

[7] Kent D. Lee and Steve Hubbard. *Data Structures and Algorithms with Python*. Springer, Cham, Switzerland, 2015.

[8] Brad N. Miller and David L. Ranum. *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle & Associates, $2^{\text{nd}}$ edition, 2011.

[9] Rance D. Necaise. *Data Structures and Algorithms Using Python*. John Wiley & Sons, New Jersey, 2011.

ISBN:978-602-361-279-6