Purwono, S.Kom., M.Kom.
Alfian Ma Arif S.T., M.Eng.,
Dr. Ir.Iswanto, S.T., M.Eng., IPM.,

Belajar Struktur Data dengan Python

Penerbit UHB Press

Belajar Struktur Data dengan Python

Purwono, S.Kom., M.Kom.
Alfian Ma Arif S.T., M.Eng.,
Dr. Ir. Iswanto, S.T., M.Eng., IPM.,

Penerbit UHB Press

Belajar Struktur Data dengan Python

Oleh:
Purwono, S.Kom., M.Kom.

Alfian Ma'arif S.T., M.Eng.,

Dr. Ir. Iswanto, S.T., M.Eng., IPM.,

Editor: Imam Ahmad Ashari, S.Kom., M.Kom.

Diterbitkan oleh **Penerbit UHB Press**

Jl. Raden Patah No.100, Ledug, Kembaran, Banyumas, Jawa Tengah, Telp. (0281) 6843493, Fax. (0281) 6843494, Purwokerto 53182

Purwono, S.Kom., M.Kom, Alfian Ma'arif S.T., M.Eng., dan Dr. Ir. Iswanto, S.T., M.Eng., IPM.,

-edisi Pertama – Purwokerto: UHB Press, 2023

xvi + - 153 hlm, 1 Jil: 23 cm

ISBN: 978-623-88102-6-0

ISBN 978-623-88102-6-0 (PDF)

9 786238 810260

Sinopsis

Dalam istilah ilmu komputer, struktur data adalah cara menyimpan dan mengatur data secara terstruktur pada sistem komputer atau pangkalan data (*database*) sehingga lebih mudah diakses. Secara teknis, data dalam bentuk angka, huruf, simbol, dan lainnya ini diletakkan dalam kolom-kolom dan susunan tertentu.

Contoh struktur data dapat dilihat pada berkas-berkas lembar sebar (*spreadsheet*), pangkalan data, pengolah kata, citra yang dipampat (*compressed image*), dan pemampatan berkas dengan teknik tertentu yang memanfaatkan struktur data.

Pada buku ini dilengkapi dengan teori dan contoh implementasi struktur data menggunakan bahasa pemrograman python. Harapannya buku ini bisa menjadi panduan dasar bagi pelajar, mahasiswa atau umum yang ingin mempelajari struktur data lebih lanjut.

Kata Pengantar

Puji syukur penulis panjatkan kepada Allah SWT yang telah memberikan kesempatan kepada penulis untuk menyusun atau merancang Buku **Belajar Struktur Data dengan Python**. Ucapan terimakasih diberikan kepada semua pihak yang telah membantu hingga penulisan buku ini dapat diselesaikan dengan tepat waktu.

Pada buku ini dilengkapi dengan dasar teori tentang struktur data meliputi tipe data primitif, *graph, list, tuple, dictionary, linked list, tree* dan sebagainya. Pada setiap bab disertai dengan kode latihan dengan bahasa pemrograman python, sehingga pembaca bisa secara langsung untuk melakukan latihan praktik secara langsung. Penulis berharap bahwa buku referensi ini dapat dipergunakan bersama dengan buku panduan lain untuk menjadi pegangan dalam mempelajari struktur data dengan pemrograman python.

Penulis menyadari masih kekurangan dan keterbatasan pada buku ini, karenanya kritik dan saran membangun sangat penulis harapkan. Besar harapan penulis agar tugas akhir ini dapat memberi manfaat bagi semua.

Purwokerto, Maret 2023

Penulis

Daftar Isi

Daftar Gambar

Daftar Tabel

Pendahuluan

Struktur data adalah alat penting untuk bahasa pemrograman apa pun, dan Python tidak terkecuali. Python menyediakan beberapa struktur data bawaan yang memungkinkan manipulasi dan pemrosesan data yang efisien. Struktur data ini digunakan untuk menyimpan, mengatur, dan memanipulasi data dalam memori.

Struktur data merupakan suatu cara mengatur dan menyimpan data agar dapat diakses dan dikerjakan dengan efisien. Mereka menentukan hubungan antara data, dan operasi yang dapat dilakukan pada data. Ada banyak jenis struktur data yang didefinisikan yang memudahkan para ilmuwan data dan insinyur komputer, sama-sama berkonsentrasi pada gambaran utama pemecahan masalah yang lebih besar daripada tersesat dalam detail deskripsi dan akses data.

Dalam teknik pemrograman, struktur data mengacu pada tata letak data yang berisi kolom data, baik kolom yang terlihat oleh pengguna atau kolom yang hanya digunakan untuk keperluan pemrograman dan tidak terlihat oleh pengguna. Setiap baris dalam daftar kolom disebut catatan. Lebar kolom data dapat berubah dan bervariasi. Ada kolom yang lebarnya berubah secara dinamis tergantung input pengguna, dan ada juga kolom yang lebarnya tetap. Struktur data secara alami dapat diterapkan pada pemrosesan basis data (misalnya untuk tujuan informasi keuangan) atau pemrosesan kata di mana kolom berubah secara dinamis.

Setiap jenis struktur data yang berbeda cocok untuk penggunaan yang berbeda, dan beberapa jenis berspesialisasi dalam tugas tertentu. Misalnya, database relasional biasanya menggunakan indeks B-tree untuk mengambil data, sedangkan implementasi kompiler biasanya menggunakan tabel hash untuk mencari tag.

Struktur data menyediakan cara untuk mengelola data dalam jumlah besar secara efisien untuk berbagai keperluan, seperti database besar dan layanan pengindeksan Internet. Secara umum, struktur data yang efisien adalah kunci untuk merancang algoritma yang efisien. Beberapa metode desain formal dan bahasa pemrograman lebih menekankan struktur data daripada algoritma sebagai faktor kunci dalam desain perangkat lunak. Struktur data dapat digunakan untuk mengelola penyimpanan dan pengambilan data yang disimpan di penyimpanan primer dan sekunder.

Memahami berbagai struktur data dan propertinya sangat penting untuk pemrograman yang efisien dengan Python. Buku ini dapat digunakan sebagai referensi untuk media pembelajaran bagi mahasiswa atau umum yang ingin mempelajari struktur data dengan bahasa pemrograman python.

BAB I
Tipe Data Primitive

1.1 Abstract Data Type and Data Structures

Saat Anda membaca di bagian pendahuluan, struktur data membantu Anda untuk fokus pada gambaran yang lebih besar daripada tersesat dalam detailnya. Ini dikenal sebagai abstraksi data.

Sekarang, struktur data sebenarnya merupakan implementasi dari Tipe Data Abstrak atau ADT. Implementasi ini membutuhkan tampilan fisik data menggunakan beberapa kumpulan konstruksi pemrograman dan tipe data dasar.

Secara umum, struktur data dapat dibagi menjadi dua kategori dalam ilmu komputer: struktur data primitif dan non-primitif. Yang pertama adalah bentuk paling sederhana untuk merepresentasikan data, sedangkan yang terakhir lebih maju: mereka berisi struktur data primitif dalam struktur data yang lebih kompleks untuk tujuan khusus. Pada modul ini kita akan memahami tipe data primitif, selanjutnya di Bab 2 akan dibahas tentang non primitif.

Perhatikan bahwa di Python, Anda tidak harus secara eksplisit menyatakan jenis variabel atau data Anda. Itu karena ini adalah bahasa yang diketik secara dinamis. Bahasa yang diketik secara dinamis adalah

bahasa di mana tipe data yang dapat disimpan objek bisa berubah (Jaiswal 2017).

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *terminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi
Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

1.2 Primitive Data Structures

Ini adalah struktur data yang paling primitif atau paling dasar. Mereka adalah blok bangunan untuk manipulasi data dan berisi nilai data yang murni dan sederhana. Python memiliki empat tipe variabel primitif:

1.2.1 Integer

Anda dapat menggunakan bilangan bulat untuk mewakili data numerik, dan lebih khusus lagi, bilangan bulat dari tak terhingga negatif hingga tak terhingga, seperti 4, 5, atau -1.

Sebagai contoh anda dapat melihat code python pada file ***integer.py*** berikut.

File **integer.py**

```
1. #integer
2. x = 1
3. y = 5
```

```
4. #jumlah antar variable
5. z = x + y
6. print(z)


7. #output 6
```

1.2.2 Float

"Float" adalah singkatan dari 'floating point number'. Anda dapat menggunakannya untuk bilangan rasional, biasanya diakhiri dengan angka desimal, seperti 1,11 atau 3,14. Sebagai contoh anda dapat melihat code python pada file ***float.py*** berikut.

File **float.py**

```
1. #float data tipe
2. x = 0.5
3. y = 0.8
4.
5. #kalikan dua variable
6. z  = x * y
7.
8. #cetak hasil
9. print(z)
10.
11. #output = 0.4
```

1.2.3 String

String adalah kumpulan huruf, kata, atau karakter lain. Di Python, Anda dapat membuat string dengan menyertakan urutan karakter dalam sepasang tanda

kutip tunggal atau ganda. Misalnya: 'sepeda', "motor", dll.

Anda juga dapat menerapkan operasi + pada dua atau lebih string untuk menggabungkannya, seperti pada contoh di bawah ini:

File **string.py**

```
1. x = "Sepeda"
2. y = "Motor"
3.
4. #gabungkan dua buah string
5. z = x + y
6.
7. #cetak string
8. #output "SepedaMotor"
9. print(z)
```

Untuk mengulang string kita dapat menggunakan tanda '*' sebagai contoh adalah sebagai berikut:

File **string.py**

```
1. #ulang string
2. x = "Sepeda"
3. a = x * 2
4. print(a)
5.
6. #output SepedaSepeda
```

Anda juga dapat memotong string, yang berarti Anda memilih bagian dari string:

File **string.py**

```
1.  #slice atau potong string
2.  x = “Sepeda”
3.  #ambil string 2 di depan
4.  b = x[:2]
5.  #ambil string setelah 2
6.  c = x[2:]
7.  #ambil string sesuai posisi
8.  #semua string dimulai dari indeks ke 0
9.  d = x[0] + x[5]
10.
11. print(b)
12. #output 'Se'
13.
14. print(c)
15. #output 'peda'
16.
17. print(d)
18. #output 'Sa'
```

Perhatikan bahwa string juga bisa berupa karakter alfanumerik, tetapi operasi + masih digunakan untuk menggabungkan string.

File **string.py**

```
1.  #string alphanumerik
2.
3.  f = '4'
4.  g = '5'
5.
6.  h = f+g
```

```
7.
8. print(h) #output "45"
```

1.2.3.1 Fungsi bawaan string

Python memiliki banyak metode bawaan atau fungsi pembantu untuk memanipulasi string. Mengganti substring, memanfaatkan kata-kata tertentu dalam paragraf, menemukan posisi string dalam string lain adalah beberapa manipulasi string yang umum. Lihat beberapa di antaranya:

Kapitalisasi string yaitu merubah awal string menjadi huruf besar.

File **string.py**

```
1. #Kapitalisasi String
2. i = 'donat'
3. j = str.capitalize(i)
4.
5. print(j) #output 'Donat'
```

Ambil panjang string dalam karakter. Perhatikan bahwa spasi juga diperhitungkan dalam hasil akhir:

File **string.py**

```
1. #hitung panjang string
2. r = "Suka Makan"
3. s = 'Hobbi'
4.
5. print(len(r)) #output 10 (spasi dihitung)
6. print(len(s)) #output 5
```

Melakukan cek apakah string tersebut hanya bernilai digit atau tidak.

File **string.py**

```
1. #cek apakah bernilai digit
2. digit = '404'
3. nondigit = 'Empat Kosong Empat'
4.
5. print(digit.isdigit()) #output True
6. print(nondigit.isdigit()) #output False
```

Mengganti bagian dari string dengan string yang lain.

File **string.py**

```
1. #mengganti bagian string dengan string lain
2. nama = 'Joko Susanto'
3. ganti = nama.replace('Joko', 'Aldi')
4.
5. print(ganti) #output 'Aldi Susanto'
```

Temukan substring di string lain; Mengembalikan indeks atau posisi terendah dalam string tempat substring ditemukan:

File **string.py**

```
1. #mencari string di string yang lain
2. menu = "Nasi Padang"
3. caristring = "Padang"
4. print(menu.find(caristring))   #output   5
```

```
karena dimulai dari index setelah 5
```

1.2.4 Boolean

Jenis data bawaan ini yang dapat mengambil nilai: True dan False, yang sering membuatnya dapat dipertukarkan dengan bilangan bulat 1 dan 0. Boolean berguna dalam ekspresi bersyarat dan perbandingan, seperti dalam contoh berikut: (Lambert 2014).

File **boolean.py**

```
1.  #Boolean
2.
3.  #Case 01
4.  harga = 1000
5.
6.  #Apakah harga lebih dari 500
7.  if harga >= 500: #kondisi
8.  #jika iya
9.      print('Diskon 10%')
10. #jika tidak
11. else:
12.     print('Tidak Dapat Diskon')
13. #output ‘Diskon 10%’
14.
15. #Case 02
16. isLogin = False
17.
18. #cek apakah isLogin == True
19. #jika iya
20. if isLogin:
21.     print('Welcome')
22. #jika tidak
```

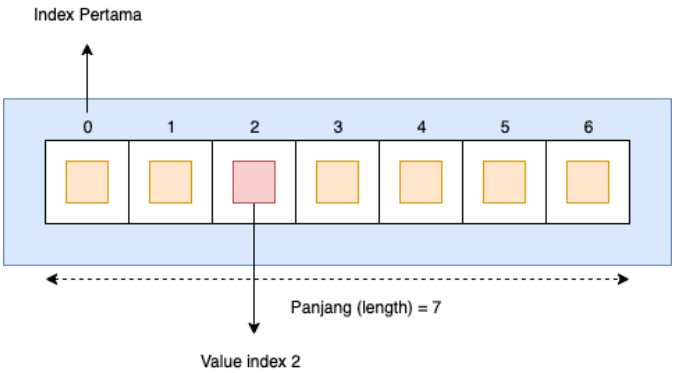
```
else:
    print('Silahkan Login')
#output ‘Silahkan Login’
```

BAB II
Array List

Ada berbagai jenis *data collections* dalam python seperti *List, Tupple, Dictionary*, dan *Set*. Pada bab ini kita akan belajar tentang struktur dasar Array terlebih dahulu. Array adalah kumpulan item yang disimpan di lokasi memori yang berdekatan. Idenya adalah untuk menyimpan beberapa item dari jenis yang sama secara bersamaan. Ini membuatnya lebih mudah untuk menghitung posisi setiap element hanya dengan menambahkan offset ke nilai dasar, yaitu lokasi memori element pertama dari array (biasanya dilambangkan dengan nama array).

Untuk kesederhanaan, kita dapat membayangkan serangkaian tangga yang pada setiap langkahnya diberi nilai (misalkan salah satu teman Anda). Di sini, Anda dapat mengidentifikasi lokasi teman Anda hanya dengan mengetahui hitungan langkah mereka. Array dapat ditangani dengan Python oleh modul bernama array. Mereka dapat berguna ketika kita harus memanipulasi hanya nilai tipe data tertentu. Seorang pengguna dapat memperlakukan daftar sebagai array. Namun, pengguna tidak dapat

membatasi jenis element yang disimpan dalam daftar. Jika Anda membuat array menggunakan modul array, semua element dari array harus berjenis sama.



Gambar 1 Array

Berdasarkan Gambar 1, sebuah array memiliki index dan dimulai dengan index ke 0, jadi semua akan dimulai dari index ke 0. Setiap index akan memiliki valuenya masing-masing. Array juga memiliki panjang valuenya.

Array adalah fondasi untuk semua ilmu data dengan Python. Array bisa multidimensi, dan semua elemen dalam array harus berjenis sama, semua integer atau semua float, misalnya. Array merupakan salah satu tipe data non primitive. Tipe non-primitif adalah anggota canggih dari keluarga struktur data. Mereka tidak hanya menyimpan nilai, melainkan kumpulan nilai dalam berbagai format.

2.1 Keuntungan menggunakan Array

Ada beberapa keuntungan yang bisa kita dapatkan saat menggunakan array. Keuntungan tersebut antara lain adalah sebagai berikut:

- Array dapat menangani kumpulan data yang sangat besar secara efisien
- Hemat memori secara komputasi
- Perhitungan dan analisis lebih cepat daripada list
- Fungsionalitas yang beragam (banyak fungsi dalam paket Python). Dengan beberapa paket Python yang membuat pemodelan tren, statistik, dan visualisasi lebih mudah.

Perhatikan bahwa di Python, Anda tidak harus secara eksplisit menyatakan jenis variabel atau data Anda. Itu karena ini adalah bahasa yang diketik secara dinamis. Bahasa yang diketik secara dinamis adalah bahasa di mana tipe data yang dapat disimpan objek bisa berubah.

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *terminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi

Pada modul ini kami menggunakan List Array sebagai bahan latihan. Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

2.2 Dasar Array

Array merupakan variabel khusus yang dapat menampung lebih dari satu nilai. Sebagai contoh saat kita memiliki daftar belanjaan yang kita simpan dengan variabel tunggal sebagai berikut:

File array-list.py.py

```
1. beli1 = 'Tomat'
2. beli2 = 'Kangkung'
3. beli3 = 'Bayam'
```

Permasalahan akan terjadi ketika ternyata kita akan membeli banyak belanjaan. Misalnya kita akan belanja super banyak dengan total 500 item, apakah kita akan secara manual menyimpan data tersebut dalam variabel tunggal? Solusi dari permasalahan tersebut dapat diatasi dengan array. Dengan array kita bisa menampung banyak nilai cukup dengan satu nama saja. Lihatlah kode berikut.

File array-list.py

```
1. beli=['Tomat','Kangkung','Bayam']
```

Jika sebelumnya kita menggunakan satu variabel untuk menampung satu data belanjaan, sekarang kita dapat menggunakan satu variabel dengan berbagai isi belanjaan.

Terdapat beberapa fungsi bawaan yang dimiliki python untuk memanipulasi data Array antara lain adalah sebagai berikut:

- append()
- pop()
- count()
- extend()
- sort()
- remove()
- insert()
- len()
- copy()
- index()
- reverse()

2.3 Akses Data Array

Dalam array kita akan terbiasa dengan index yaitu posisi sebuah value dari array tersebut. Misalnya pada index array ke 5 ada sebuah value ‘Jeruk’. Berikut ini merupakan cara kita dapat mengakses sebuah element value di array python.

File **array-list.py**

```
1. beli = ['Tomat','Kangkung','Bayam']
2.
3. cetak = beli[0] #ambil value di index ke 0
4.
5. print(cetak) #output 'Tomat'
```

Pada code di atas, sebuah variabel array dengan nama ‘beli’ memiliki tiga buah value yaitu Tomat, Kangkung dan Bayam. Karena array menggunakan cara indeksing dan dimulai dari indeks ke-0, kita dapat dengan mudah mencari value dari posisi indeks tersebut. Dalam array beli Tomat berada di indeks ke-0, Kangkung di indeks ke-1 dan Bayam di indeks ke-2. Pada baris ke-3 sebuah variabel dengan nama cetak dia bernilai array beli yang diposisikan pada indeks ke-0 sehingga ketika kita cetak dengan fungsi print, data element nilai yang di hasilkan adalah ‘Tomat’.

Cara lain untuk mengakses element value dari sebuah array dapat menggunakan perulangan seperti for. Lihat kode berikut ini.

File **array-list.py**

```
1. beli = ['Tomat','Kangkung','Bayam']
2.
3. #perulangan untuk akses array
4. for x in beli:
5.     #tercetak semua element value = Tomat,
   Kangkung, Bayam
6.     print(x)
```

2.4 len() - Panjang Array

Panjang dari suatu array bisa kita dapatkan informasinya. Python sudah memberikan fungsi bawaan yaitu **len**. Untuk lebih jelasnya silahkan lihat code berikut.

File **array-list.py**

```
1. beli=['Tomat','Kangkung','Bayam']
2. panjang = len(beli)
3. print(panjang) #output 3
```

2.5 append() - Menambahkan Element Array

Metode untuk menambahkan nilai element tententu pada array dapat menggunakan fungsi **append**. Sebagai contoh kita memiliki variabel array dengan nilai data adalah kosong. Kita bisa menambahkan dengan append value yang kita inginkan. Lihat kode berikut.

File **array-list.py**

```
1. #append
2. #menambahkan element array
3.
4. ambil = []
5. ambil.append('Mangga')
6. ambil.append('Jeruk')
7.
8. print(ambil) #output ['Mangga', 'Jeruk']
```

2.6 pop() - Menghapus Element Array

Metode untuk menghapus element array tertentu dapat menggunakan fungsi **pop**. Fungsi ini akan meminta parameter index array yang akan dihapus. Sebagai contoh kita memiliki array sayur yang

memiliki 4 buah element, kemudian kita menghapus salah satunya berdasarkan index.

File **array-list.py**

```
1. #remove element array
2. sayur = ['Kol','Wortel','Buncis','Toge']
3. #remove element di index ke 1
4. sayur.pop(1)
5.
6. print(sayur) #output
   ['Kol','Buncis','Toge']
```

Selain dengan **pop** kita juga bisa spesifik menghapus element array dengan memastikan value yang hendak dihapus. Sebagai contoh saya akan menghapus element ‘Buncis’ pada array sayur.

File **array-list.py**

```
1. #remove array element
2. sayur.remove('Buncis')
3. print(sayur) #output ['Kol', 'Toge']
```

2.7 clear() - Menghapus Semua Element Array

Metode untuk menghapus semua element yang terdapat pada array dapat menggunakan **clear**.

File **array-list.py**

```
1. #remove all array element
2. bunga = ['Melati','Mawar','Anggrek']
3.
4. #remove all
5. bunga.clear() #menghapus semua element
6.
7. print(bunga) #output []
```

2.8 copy() - Menyalin Semua Element Array

Metode untuk menyalin semua element yang terdapat pada array dapat menggunakan **copy**. Anda dapat menyalinnya pada variable baru.

File **array-list.py**

```
1. #copy array
2. motor = ['Yamaha','Honda','Suzuki']
```

```
3.  #copy to motor2
4.  motor2 = motor.copy()
5.
6.  print(motor2) #output
    ['Yamaha','Honda','Suzuki']
```

2.9 count() - Menghitung Jumlah Element Array Tertentu

Metode untuk mengetahui element array tertentu dapat dengan menggunakan fungsi **count**. Sebagai contoh anda ingin mendeteksi ada berapa element tertentu dalam sebuah array. Fungsi ini akan mengembalikan nilai jumlah element.

File **array-list.py**

```
1.  #count array
7.  mahasiswa =
    ['Harry','John','Maya','Brisya','John']
2.
3.  #count
4.  jumlah = mahasiswa.count('John')
5.
6.  print(jumlah) #output 2
```

2.10 extend() - Menambahkan list element array pada akhir list array saat ini

Metode untuk menambahkan list array pada posisi array saat ini dapat menggunakan fungsi **extend**. Fungsi ini seolah akan menggabungkan dua buah array namun digabungkan pada akhir list array saat ini.

File **array-list.py**

```
1. #extend array
2. siswa = ['Juki','Ahmad','Roni']
3. siswabaru = ['Maya','Ari','Aji']
4.
5. siswa.extend(siswabaru)
6. print(siswa) #output ['Juki', 'Ahmad',
   'Roni', 'Maya', 'Ari', 'Aji']
```

2.11 index() – Mengembalikan posisi index element array

Metode untuk mengetahui posisi index element array dapat menggunakan fungsi index. Ingat bahwa dalam array index dimulai dari posisi ke-0.

File **array-list.py**

```
1. #index array element
2. kota =
   ['Jakarta','Bandung','Surabaya','Makasar']
3.
4. #cek posisi element Bandung
5. x = kota.index('Bandung')
6.
7. print(x) #output 1
```

2.12 insert() – Menambahkan element array pada posisi tertentu secara spesifik

Metode untuk menambahkan element array pada posisi tertentu secara spesifik. Sebagai contoh anda ingin menambahkan element pada posisi ke-3 dan sebagainya.

File **array-list.py**

```
1. #insert array specific position
2. fruits = ['apple', 'banana', 'cherry']
3.
4. #insert element at index 1
5. fruits.insert(1, "orange")
```

```
6.
7.  print(fruits) #output ['apple', 'orange',
    'banana', 'cherry']
```

2.13 reverse() – Mengembalikan posisi order element array

Metode untuk mengembalikan posisi order element array dapat menggunakan fungsi reverse(). Pada fungsi ini index ke-0 akan menjadi index array terakhir.

File **array-list.py**

```
1.  #reverse
2.  angka = [1,2,3,4,5]
3.
4.  angka.reverse()
5.
6.  print(angka) #output [5, 4, 3, 2, 1]
```

2.14 sort() – Mengurutkan list array

Metode untuk mengurutkan array non digit dengan metode ascending (a-z). Sedangkan untuk angka maka akan diurutkan pada posisi element terkecil.

File **array-list.py**

```
1.  #sort array
2.  cars = ['Ford', 'BMW', 'Volvo']
3.
4.  angka = [1,6,7,3,2,5]
5.
6.  cars.sort()
7.  angka.sort()
8.
9.  print(cars) #output ['BMW', 'Ford',
    'Volvo']
10. print(angka) #output [1, 2, 3, 5, 6, 7]
```

BAB III
Array, List, Tupple, Dictionary, Set

Pada bab sebelumnya kita telah belajar tentang dasar penggunaan *Array* yang secara umum menggunakan *array list*. Modul ini akan menjelaskan perbedaan secara umum antara array dan list pada bahasa pemrograman python. Baik list dan array digunakan untuk menyimpan data. Selain itu, kedua struktur data memungkinkan pengindeksan (*index*), pemotongan (*slice*), dan pengulangan (*loop*). Jadi apa perbedaan antara array dan list dengan Python? Dalam modul ini, kami akan menjelaskan secara detail kapan harus menggunakan *array* dan *list*.

Python memiliki banyak struktur data yang berbeda dengan fitur dan fungsi yang berbeda. Struktur data bawaannya mencakup *list, tupple, set*, dan *dictionary*. Namun, ini bukanlah daftar lengkap dari struktur data yang tersedia dengan Python. Beberapa struktur data tambahan dapat diimpor dari modul atau paket yang berbeda.

Struktur data *array* termasuk dalam kategori "harus di impor" sehingga anda perlu menginstalnya dengan *python installer package manager* (pip). Untuk

menggunakan array dengan Python, Anda perlu mengimpor struktur data ini dari paket *NumPy* atau modul *array*.

Dan itulah perbedaan pertama antara list dan array. Sebelum menyelami lebih dalam perbedaan antara dua struktur data ini, mari kita tinjau fitur dan fungsi *list* dan *array*.

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *teminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi
Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

3.1 List

List adalah struktur data yang dibangun ke dalam Python dan menampung sekumpulan item. *List* memiliki sejumlah karakteristik penting:

- Item *list* diapit dalam tanda kurung siku, seperti ini [item1, item2, item3].
- *List* diurutkan - yaitu item dalam daftar muncul dalam urutan tertentu. Ini memungkinkan kita menggunakan indeks

untuk mengakses item apa pun.

- *List* bisa berubah, yang berarti Anda dapat menambah atau menghapus item setelah pembuatan list.
- Element *list* tidak harus unik. Duplikasi item dimungkinkan, karena setiap element memiliki tempat yang berbeda dan dapat diakses secara terpisah melalui indeks.
- Element bisa dari tipe data yang berbeda: Anda bisa menggabungkan *string, integer*, dan *objek* dalam daftar yang sama.

Pada code **list-array.py** anda akan melihat penggunaan *list*.

File **list-array.py**

```
1.  #List python
2.
3.  listangka = [3, 6, 9, 12]
4.  listhuruf = ['a','b','c','d']
5.  listrandom = [3,'a', 6, True, 12]
6.
7.  print(listangka) #output [3, 6, 9, 12]
8.  print(listhuruf) #output ['a', 'b', 'c',
    'd']
9.  print(listrandom) #output 3, 'a', 6, True,
    12]
10.
```

```
11. print(type(listangka))    #output    <class
    'list'>
12. print(type(listhuruf))    #output    <class
    'list'>
13. print(type(listrandom))   #output    <class
    'list'>
```

Anda telah mempelajari lebih dalam pada penggunaan *list* di modul dasar *array*. Jika anda belum mempelajarinya lebih anda terlebih dahulu membuka modul tersebut.

3.2 Array

Array juga merupakan struktur data yang menyimpan sekumpulan item. Seperti *list*, *array* diurutkan, dapat diubah, diapit dalam tanda kurung siku, dan dapat menyimpan item non-unik.

Tetapi ketika berbicara tentang kemampuan *array* untuk menyimpan tipe data yang berbeda, jawabannya tidak sesederhana itu. Itu tergantung pada jenis *array* yang digunakan.

Untuk menggunakan *array* dengan Python, Anda perlu mengimpor modul *array* atau paket NumPy.

File **array-mode.py**

```
1. #menggunakan array khusus
2. import array as arr
3. import numpy as np
```

Modul *array* Python mengharuskan semua element *array* memiliki tipe yang sama. Selain itu, untuk membuat *array*, Anda harus menentukan tipe nilai. Pada kode di bawah ini, "i" menandakan bahwa semua element dalam ***array_1*** adalah bilangan bulat:

File **array-mode.py**

```
1. #menggunakan array khusus
2. import array as arr
3.
4. array_1 = arr.array("i", [3, 6, 9, 12])
   #buat array tipe integer
5. print(array_1) #output array('i', [3, 6,
   9, 12])
6. print(type(array_1)) #output <class
   'array.array'>
```

3.2.1 Membuat Sebuah Array

Array dengan Python dapat dibuat dengan mengimpor modul *array*. *array (data_type, value_list)* digunakan untuk membuat *array* dengan tipe data dan daftar nilai yang ditentukan dalam argumennya.

File **array-new.py**

```
1.  # Membuat Array Baru
2.
3.  # importing "array" untuk array baru
4.  import array as arr
5.
6.  # membuat array dengan tipe integer (i)
7.  a = arr.array('i', [1, 2, 3])
8.
9.  # cetak data array
10. print ("Array baru yang dibuat adalah: ",
    end =" ")
11. for i in range (0, 3):
12.     print (a[i], end =" ")
13. print()
```

Beberapa tipe data disebutkan di bawah ini yang akan membantu dalam membuat array tipe data yang berbeda.

Tabel 1 Tipe data

Type Code	**C Type**	**Python Type**	**Minimum Size in Bytes**
‘b’	Signed char	int	1
‘B’	Unsigned char	int	1
‘u’	Py_UNICODE	Unicode character	2
‘h’	Signed short	int	2

‘H’	Unsigned short	int	2
‘i’	Signed int	int	2
‘I’	Unsigned int	int	2
‘l’	Signed long	int	4
‘L’	Unsigned long	int	4
‘q’	Singned long long	int	8
‘Q’	Unsigned long long	int	8
‘f’	float	float	4
‘d’	double	float	8

3.2.2 Menambahkan Element ke Sebuah Array

Element dapat ditambahkan ke *Array* dengan menggunakan fungsi *insert ()* bawaan. *Insert* digunakan untuk memasukkan satu atau lebih element data ke dalam *array*. Berdasarkan persyaratan, element baru dapat ditambahkan di awal, akhir, atau indeks larik apa pun. *append ()* juga digunakan untuk menambahkan nilai yang disebutkan dalam argumennya di akhir *array*.

File **array-append.py**

```
1. # importing "array" untuk membuat array
2. import array as arr
3.
```

```
4.  # array dengan tipe integer
5.  a = arr.array('i', [1, 2, 3])
6.
7.
8.  print ("Array Integer sebelum dilakukan
    insert : ", end =" ")
9.  for i in range (0, 3):
10.     print (a[i], end =" ")
11. print()
12.
13. # menambahkan elemen dengan
14. # fungsi insert
15. a.insert(1, 4)
16.
17. print ("Array Integer setelah dilakukan
    insert : ", end =" ")
18. for i in (a):
19.     print (i, end =" ")
20. print()
21.
22. # array dengan tipe float
23. b = arr.array('d', [2.5, 3.2, 3.3])
24.
25. print ("Array Float sebelum dilakukan
    insert : ", end =" ")
26. for i in range (0, 3):
27.     print (b[i], end =" ")
28. print()
29.
30. # menambahkan element dengan append
31. b.append(4.4)
32.
33. print ("Array Float setelah dilakukan
    insert : ", end =" ")
```

```
for i in (b):
    print (i, end =" ")
print()
```

3.2.3 Mengakses Element pada Sebuah Array

Untuk mengakses item *array*, lihat nomor indeks. Gunakan operator indeks [] untuk mengakses item dalam larik. Indeks harus berupa bilangan bulat.

File **array-access.py**

```
# importing array module
import array as arr

# array dengan tipe integer
a = arr.array('i', [1, 2, 3, 4, 5, 6])

# mengakses elemen sebuah array
print("Akses element index ke 0: ", a[0])

# accessing element of array
print("Akses element index ke 3: ", a[3])

# array dengan tipe float
b = arr.array('d', [2.5, 3.2, 3.3])

# mengakses elemen sebuah array
print("Akses element index ke 1: ", b[1])

# mengakses elemen sebuah array
print("Akses element index ke 2: ", b[2])
```

3.2.4 Menghapus Element pada Sebuah Array

Element dapat dihapus dari *array* dengan menggunakan fungsi *remove ()* bawaan tetapi Kesalahan muncul jika element tidak ada di *set*. Metode *remove ()* hanya menghapus satu element pada satu waktu, untuk menghapus berbagai element, iterator digunakan. fungsi *pop ()* juga dapat digunakan untuk menghapus dan mengembalikan element dari *array*, tetapi secara default hanya menghapus element terakhir dari *array*, untuk menghapus element dari posisi tertentu dari *array*, indeks element dilewatkan sebagai argumen ke metode *pop ()*.

Catatan – metode *Remove* dalam *List* hanya akan menghapus kemunculan pertama dari element yang dicari.

File **array-remove.py**

```
1.  # importing "array" module
2.  import array
3.
4.  # inisialisasi array
5.  # array dengan tipe integer
6.  arr = array.array('i', [1, 2, 3, 1, 5])
7.
8.  # cetak array pertama
9.  print ("Array pertama : ", end ="")
10. for i in range (0, 5):
```

```
    print (arr[i], end =" ")

print ("\r")

# menggunakan fungsi pop() untuk menghapus elemen pada posisi ke index ke-2
print ("Element yang dihapus adalah : ", end ="")
print (arr.pop(2))

# cetak array telah dilakukan pop()
print ("Array setelah di pop() : ", end ="")
for i in range (0, 4):
    print (arr[i], end =" ")

print("\r")

# Menggunakan remove() untuk menghapus yang pertama
arr.remove(1)

# cetak array setelah dihapus dengan remove()
print ("Array setelah proses remove ", end ="")
for i in range (0, 3):
    print (arr[i], end =" ")
```

3.2.5 Memotong Sebuah Array

Dalam *array Python*, ada banyak cara untuk mencetak seluruh *array* dengan semua element, tetapi

untuk mencetak berbagai element tertentu dari *array*, kami menggunakan operasi *Slice*. Operasi *slice* dilakukan pada larik dengan menggunakan titik dua (:). Untuk mencetak element dari awal hingga range tertentu, gunakan [: Index], untuk mencetak element dari penggunaan akhir [: -Index], untuk mencetak elemen dari *Index* tertentu hingga akhir gunakan [Index:], untuk mencetak element dalam rentang, gunakan [Indeks Awal: Indeks Akhir] dan untuk mencetak seluruh List dengan menggunakan operasi *slice*, gunakan [:]. Selanjutnya, untuk mencetak seluruh *array* dalam urutan terbalik, gunakan [:: - 1].

File **array-slice.py**

```
1.  # importing array module
2.  import array as arr
3.
4.  # membuat sebuah list
5.  l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
6.
7.  a = arr.array('i', l)
8.  print("Intial Array: ")
9.  for i in (a):
10.     print(i, end =" ")
11.
12. # cetak element dalam range tertentu
13. # Gunakan operasi slice
14. Sliced_array = a[3:8]
15. print("\nAmbil semua elemen pada range 3-
    8: ")
```

```
print(Sliced_array)

# Cetak elemen pada
# titik yang telah ditentukan ke akhir
Sliced_array = a[5:]
print("\nElemen dipotong mulai dari posisi
    5 "
      "hingga elemen terakhir: ")
print(Sliced_array)

# Cetak elemen pada
# mulai sampai akhir
Sliced_array = a[:]
print("\nCetak semua element dengan
    operasi slice: ")
print(Sliced_array)
```

3.2.6 Mencari Element pada Sebuah Array

Untuk mencari element dalam *array* kita menggunakan metode *python in-built index ()*. Fungsi ini mengembalikan indeks kemunculan pertama dari nilai yang disebutkan dalam argumen.

File **array-find.py**

```
# importing array module
import array

# siapkan array integer
arr = array.array('i', [1, 2, 3, 1, 2, 5])

# cetak array asli
```

```
8.  print ("Array asli adalah : ", end ="")
9.  for i in range (0, 6):
10.     print (arr[i], end =" ")
11.
12. print ("\r")
13.
14. # Gunakan index() untuk mencetak element
    dengan nilai 2
15. print ("Index array untuk element dengan
    nilai 2 adalah : ", end ="")
16. print (arr.index(2))
17.
18. # # Gunakan index() untuk mencetak element
    dengan nilai 1
19. print ("Index array untuk element dengan
    nilai 2 adalah ", end ="")
20. print (arr.index(1))
```

3.2.7 Mengupdate Element pada Sebuah Array

Untuk memperbarui element dalam *array*, kami cukup menetapkan ulang nilai baru ke indeks yang diinginkan yang ingin kami perbarui.

File **array-update.py**

```
1.  # importing array module
2.  import array
3.
4.  # siapkan data array integer
5.  arr = array.array('i', [1, 2, 3, 1, 2, 5])
6.
7.  # cetak array asli
```

```
print ("Array sebelum update : ", end ="")
for i in range (0, 6):
    print (arr[i], end =" ")

print ("\r")

# Update array index ke 2 dengan nilai
  element 6
arr[2] = 6
print("Array setelah update : ", end ="")
for i in range (0, 6):
    print (arr[i], end =" ")
print()

# Update array index ke 4 dengan nilai
  elemen 8
arr[4] = 8
print("Array setelah update : ", end ="")
for i in range (0, 6):
    print (arr[i], end =" ")
```

3.3 Perbedaan Array dan List

Berikut ini adalah perbedaan mendasar antara *Array* dan *List* pada Python:

- *Array* perlu dideklarasikan. *List* tidak, karena dibuat ke dalam Python. Pada contoh di atas, Anda melihat bahwa daftar dibuat hanya dengan melampirkan urutan element ke dalam tanda kurung siku. Sebaliknya, membuat

sebuah *array* membutuhkan fungsi tertentu dari modul *array* (yaitu, array.array ()) atau paket NumPy (misalnya, *numpy.array ()*). Karena itu, *list* lebih sering digunakan daripada *array*.

- *Array* dapat menyimpan data dengan sangat kompak dan lebih efisien untuk menyimpan data dalam jumlah besar.
- *Array* sangat bagus untuk operasi numerik; *list* tidak bisa langsung menangani operasi matematika. Misalnya, Anda dapat membagi setiap element *array* dengan nomor yang sama hanya dengan satu baris kode.

File **array-mode.py**

```
1. array = np.array([3, 6, 9, 12])
2. division = array/3
3. print(division) #output [1. 2. 3. 4.]
4. print (type(division)) #output <class
   'numpy.ndarray'>
5.
6. list = [3, 6, 9, 12]
7. division = list/3 #error unsupported
   operand type(s) for /: 'list' and 'int'
```

3.4 Tuple

Tuple dalam Python mirip dengan list. Perbedaan antara keduanya adalah kita tidak dapat mengubah elemen tuple setelah ditugaskan sedangkan kita dapat mengubah elemen list. Tuple digunakan untuk menyimpan banyak item dalam satu variabel.

Tuple juga merupakan salah satu dari 4 tipe data bawaan di Python yang digunakan untuk menyimpan kumpulan data, 3 lainnya adalah List, Set, dan Dictionary, semuanya dengan kualitas dan penggunaan yang berbeda.

Tuple ialah koleksi yang bersifat ordered yaitu ketika kita mengatakan bahwa tupel di order, itu berarti item memiliki order yang ditentukan, dan pesanan itu tidak akan berubah. Tuple tidak dapat diubah, artinya kita tidak dapat mengubah, menambah atau menghapus item setelah tupel dibuat. Tuple ditulis dengan tanda kurung bulat.

3.4.1 Keuntungan Sebuah Tuple

Karena tupel sangat mirip dengan list, keduanya digunakan dalam situasi yang serupa. Namun, ada beberapa keuntungan menerapkan tuple di atas list yaitu:

- Kita biasanya menggunakan tupel untuk tipe data heterogen (berbeda) dan list untuk tipe data homogen (mirip).
- Karena tuple tidak dapat diubah, iterasi melalui tuple lebih cepat dibandingkan dengan list. Jadi ada sedikit peningkatan kinerja.
- Tupel yang berisi elemen yang tidak dapat diubah dapat digunakan sebagai kunci kamus. Dengan list, ini tidak mungkin.
- Jika kita memiliki data yang tidak berubah, mengimplementasikannya sebagai tuple akan menjamin bahwa data tersebut tetap terproteksi.

3.4.2 Membuat Sebuah Tuple

Tuple dibuat dengan menempatkan semua item (elemen) di dalam tanda kurung (), dipisahkan dengan koma. Tanda kurung bersifat opsional, namun merupakan praktik yang baik untuk menggunakannya.

Tuple dapat memiliki sejumlah item dan mungkin dari tipe yang berbeda (integer, float, list, string, dll.).

File **create-tuple.py**

```
# Berbagai jenis tupel

# tuple kosong
my_tuple = ()
print(my_tuple) #output ()

# Tuple dengan isi integers
my_tuple = (1, 2, 3)
print(my_tuple) #output (1, 2, 3)

# tuple dengan campuran datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple) #output (1, 'Hello', 3.4)

# tuple bersarang
my_tuple = ("python", [8, 4, 6], (1, 2, 3))
print(my_tuple) #output ('python', [8, 4, 6], (1, 2, 3))
```

Dalam contoh di atas, kita telah membuat berbagai jenis tupel dan menyimpan berbagai item data di dalamnya. Seperti disebutkan sebelumnya, kita juga bisa membuat tupel tanpa menggunakan tanda kurung.

File **create-tuple-tanpa-kurung.py**

```
my_tuple = 1, 2, 3
print(my_tuple) #output (1, 2, 3)
my_tuple = 1, "Hello", 3.4
```

```
4. print(my_tuple) #output (1, 'Hello', 3.4)
```

3.4.3 Membuat Tuple dengan Satu Elemen

Di Python, membuat tuple dengan satu elemen dapat dibilang cukup rumit. Memiliki satu elemen dalam tanda kurung tidaklah cukup, oleh karena itu kita akan membutuhkan tanda koma untuk menunjukkan bahwa itu adalah tuple.

File **create-tuple.py**

```
1. var1 = ("Hello") # string
2. var2 = ("Hello",) # tuple
```

Kita dapat menggunakan fungsi type() untuk mengetahui di kelas mana variabel atau nilai berada.

File **create-tuple.py**

```
1. var1 = ("hello")
2. print(type(var1))  # <class 'str'>
3.
4. # membuat sebuah tuple dengan satu element
5. var2 = ("hello",)
6. print(type(var2))  # <class 'tuple'>
7.
8. # Tanda kurung adalah opsional
9. var3 = "hello",
10. print(type(var3))  # <class 'tuple'>
```

Di Sini, ("hello") adalah sebuah string sehingga type() mengembalikan str sebagai kelas var1 yaitu <class 'str'> sedangkan ("halo",) dan "halo", keduanya tuple jadi type() mengembalikan tuple sebagai kelas var1 yaitu <class 'tuple'>.

3.4.4 Akses Elemen Tuple

Seperti list, setiap elemen tupel diwakili oleh nomor indeks (0, 1, ...) di mana elemen pertama berada di indeks 0. Kita menggunakan nomor indeks untuk mengakses elemen tuple. Terapat beberapa cara untuk mengakses elemen tupel yaitu

1. **Indexing**

   Kita dapat menggunakan operator indeks [] untuk mengakses item dalam sebuah tuple, dimana indeks dimulai dari 0. Jadi, tuple yang memiliki 6 elemen akan memiliki indeks dari 0 hingga 5. Mencoba mengakses indeks di luar rentang indeks tuple (6,7,... dalam contoh ini) akan memunculkan IndexError.

   Indeks harus berupa bilangan bulat, jadi kita tidak bisa menggunakan float atau tipe lainnya. Ini akan menghasilkan TypeError.

   Demikian pula, tupel bersarang diakses menggunakan pengindeksan bersarang,

seperti yang ditunjukkan pada contoh di bawah ini.

File **tuple.py**

```
1. # mengakses tuple dengan teknik indeksing
2. letters = ("p", "r", "o", "g", "r", "a",
   "m", "i", "z")
3.
4. print(letters[0])   # prints "p"
5. print(letters[5])   # prints "a"
```

Dalam contoh di atas, letters[0] - mengakses elemen pertama dan letters[5] - mengakses elemen keenam.

2. Negative Indexing

   Python memungkinkan pengindeksan negatif untuk urutannya. Indeks -1 merujuk ke item terakhir, -2 ke item terakhir kedua dan seterusnya. Kita dapat melihat contoh berikut.

File **tuple.py**

```
1. # mengakses tuple dengan indeks negatif
2. letters = ('p', 'r', 'o', 'g', 'r', 'a',
   'm', 'i', 'z')
3.
4. print(letters[-1])   # prints 'z'
5. print(letters[-3])   # prints 'm'
```

Dalam contoh di atas, letters[-1] - mengakses elemen terakhir dan letters[-3] - mengakses elemen terakhir ketiga.

3. Slicing

   Kita dapat mengakses berbagai item dalam sebuah tuple dengan menggunakan titik dua operator pengiris : (simbol titik dua). Kita dapat melihat cara penggunaannya pada kode berikut.

File **tuple.py**

```
1. # mengakses tuple elemen dengan teknik
   slicing / irisan
2. my_tuple = ('p', 'r', 'o', 'g', 'r', 'a',
   'm', 'i', 'z')
3.
4. # element pada indeks ke 2 hingga ke 4
5. print(my_tuple[1:4])  #  prints ('r', 'o',
   'g')
6.
7. # element dua dari awal
8. print(my_tuple[:-7]) # prints ('p', 'r')
9.
10. # element ke 8 hingga akhir
11. print(my_tuple[7:]) # prints ('i', 'z')
12.
13. # elem
14. print(my_tuple[:]) # Prints ('p', 'r',
    'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Berdasarkan kode di atas, my_tuple[1:4] mengembalikan tuple dengan elemen dari indeks 1 hingga indeks 3. my_tuple[:-7] mengembalikan tuple dengan elemen dari awal hingga indeks 2. my_tuple[7:] mengembalikan tuple dengan elemen dari indeks 7 sampai akhir. my_tuple[:] mengembalikan semua item tuple.

Catatan: Saat kami melakukan slice pada list, indeks awal bersifat inklusif tetapi indeks akhir bersifat eksklusif.

3.4.5 Metode pada Tuple

Di Python, metode yang menambah item atau menghapus item tidak tersedia dengan tuple. Hanya terdapat dua metode yang tersedia. Berikut ini adalah contoh metode yang bisa digunakan.

File **tuple.py**

```
1. my_tuple = ('a', 'p', 'p', 'l', 'e',)
2.
3. print(my_tuple.count('p')) # prints 2
4. print(my_tuple.index('l')) # prints 3
```

Berdasarkan kode di atas maka, my_tuple.count(‘p’) digunakan untuk menghitung jumlah total ‘p;’di

my_tuple dan my_tuple.index('l') digunakan untuk mengembalikan kemunculan pertama 'l' di my_tuple.

1. Iterasi melalui Tuple

   Kita dapat menggunakan perulangan for untuk mengulangi elemen tupel.

File **tuple.py**

```
1. languages = ('Python', 'Swift', 'C++')
2.
3. # iterasi melalui tuple
4. for language in languages:
5.     print(language)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Python
Swift
C++
```

2. Memeriksa apakah Item Ada di Tuple

   Kita dapat menggunakan kata kunci **in** untuk memeriksa apakah suatu item ada di tuple atau tidak.

File **tuple.py**

```
1. languages = ('Python', 'Swift', 'C++')
2.
3. print('C' in languages) # False
```

```
4. print('Python' in languages) # True
```

Berdasarkan kode di atas maka, 'C' tidak ada dalam **languages**, 'C' dalam **languages** dievaluasi menjadi **False**. 'Python' hadir dalam **languages**, 'Python' dalam **language** dievaluasi menjadi **True**.

3.5 Dictionary

Dictionary dalam Python adalah kumpulan nilai kunci, digunakan untuk menyimpan nilai data seperti map, yang tidak seperti tipe data lain yang hanya menyimpan satu nilai sebagai elemen. Dictionary memegang pasangan key:value. Key-Value disediakan dalam dictionary untuk membuatnya lebih optimal.

File **dictionary.py**

```
1. Dict = {1: 'Apple', 2: 'For', 3: 'A'}
2. print(Dict) #output {1: 'Apple', 2: 'For',
   3: 'A'}
```

3.5.1 Membuat Dictionary

Di Python, dictionary bisa dibuat dengan menempatkan urutan elemen di dalam kurung kurawal {}, dipisahkan dengan 'koma'. Dictionary menyimpan pasangan nilai, satu menjadi Kunci (Key)

dan elemen pasangan yang sesuai lainnya menjadi Nilainya (Value). Nilai dalam dictionary dapat berupa tipe data apa pun dan dapat digandakan, sedangkan kunci tidak dapat diulang dan harus tidak dapat diubah. Kunci dictionary peka terhadap huruf besar/kecil, nama yang sama tetapi huruf Kunci yang berbeda akan diperlakukan secara berbeda.

File **dictionary.py**

```
1. Dict = {1: 'Apple', 2: 'For', 3: 'A'}
2. print("\nDictionary dengan penggunaan keys
   integer:: ")
3. print(Dict)
4.
5. # Buat sebuah Dictionary
6. # dengan campuran keys (string dan
   integer)
7. Dict = {'Name': 'Apple', 1: [1, 2, 3, 4]}
8. print("\nDictionary dengan keys campuran:
   ")
9. print(Dict)
```

Hasil jika kode tersebut di run adalah sebagai berikut:

```
Dictionary dengan penggunaan keys integer::
{1: 'Apple', 2: 'For', 3: 'A'}

Dictionary dengan keys campuran: {'Name':
'Apple', 1: [1, 2, 3, 4]}
```

Dictionary juga dapat dibuat dengan fungsi built-in dict(). Dictionary kosong bisa dibuat hanya dengan menempatkan kurung kurawal{}.

File **dictionary.py**

```
1.  # membuat dictionary kosong
2.  Dict = {}
3.  print("Empty Dictionary: ")
4.  print(Dict)
5.
6.  # Membuat sebuah Dictionary
7.  # dengan dict() method
8.  Dict = dict({1: 'Apple', 2: 'For', 3:
    'A'})
9.  print("\nDictionary dengan penggunaan
    dict(): ")
10. print(Dict)
11.
12. # Membuat sebuah Dictionary
13. # dengan setiap item sebagai Pasangan
14. Dict = dict([(1, 'Apple'), (2, 'For')])
15. print("\nDictionary dengan setiap item
    berpasangan: ")
16. print(Dict)
```

Hasil code python jika dijalankan adalah sebagai berikut:

```
Empty Dictionary:
{}
```

```
Dictionary dengan penggunaan dict():
{1: 'Apple', 2: 'For', 3: 'A'}

Dictionary dengan setiap item berpasangan:
{1: 'Apple', 2: 'For'}
```

Kita juga bisa membuat nested dictionary atau dictionary bersarang. Berikut adalah contoh penerapannya.

File **nested-dictionary.py**

```
1. # Membuat sebuah Dictionary bersarang
2. Dict = {1: 'Apple', 2: 'For',
3.          3: {'A': 'One', 'B': 'Two', 'C':
   'Three'}}
4.
5. print(Dict)
```

Hasil dari eksekusi kode tersebut adalah sebagai berikut.

```
{1: 'Apple', 2: 'For', 3: {'A': 'One', 'B':
'Two', 'C': 'Three'}}
```

3.5.2 Menambahkan Elemen pada Dictionary

Penambahan elemen dapat dilakukan dengan berbagai cara. Satu nilai pada satu waktu dapat ditambahkan ke Dictionary dengan menentukan

value beserta key-nya, mis. Dict[Key] = ‘Value’. Memperbarui value yang ada dalam Dictionary dapat dilakukan dengan menggunakan metode update() bawaan. Nilai nested key juga dapat ditambahkan ke Dictionary yang sudah ada. Saat menambahkan value, jika value sebuah key sudah ada, value akan diperbarui jika tidak, Key baru dengan value ditambahkan ke Dictionary. Berikut ini adalah contoh dari penggunaan dictionary untuk menambahkan elemen tertentu.

File **dictionary.py**

```
1.  # Membuat sebuah Dictionary kosong
2.  Dict = {}
3.  print("Empty Dictionary: ")
4.  print(Dict)
5.
6.  # Menambahkan elemen dalam satu waktu
7.  Dict[0] = 'Apple'
8.  Dict[2] = 'For'
9.  Dict[3] = 1
10. print("\nDictionary setelah menambahkan 3
    elements: ")
11. print(Dict)
12.
13. # Menambahkan set nilai
14. # ke satu Kunci
15. Dict['Value_set'] = 2, 3, 4
16. print("\nDictionary setelah menambahkan 3
    element: ")
```

```
print(Dict)

# Mengupdate key value yang ada
Dict[2] = 'Welcome'
print("\nKey value yang diupdate: ")
print(Dict)

# Menambahkan nilai Nested Key ke
    Dictionary
Dict[5] = {'Nested': {'1': 'Life', '2':
    'Balance'}}
print("\nMenambahkan sebuah Nested Key: ")
print(Dict)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Empty Dictionary:
{}
Dictionary setelah menambahkan 3 elements:
{0: 'Apple', 2: 'For', 3: 1}
Dictionary setelah menambahkan 3 element:
{0: 'Apple', 2: 'For', 3: 1, 'Value_set': (2,
3, 4)}
Key value yang diupdate:
{0: 'Apple', 2: 'Welcome', 3: 1, 'Value_set':
(2, 3, 4)}
Menambahkan sebuah Nested Key:
{0: 'Apple', 2: 'Welcome', 3: 1, 'Value_set':
(2, 3, 4), 5:
{'Nested': {'1': 'Life', '2': 'Balance'}}}
```

3.5.3 Mengakses Elemen pada Dictionary

Di Python, untuk mengakses item Dictionary merujuk ke nama kuncinya. Kunci dapat digunakan di dalam tanda kurung siku.

File **dictionary.py**

```
1.  # Membuat sebuah Dictionary
2.  Dict = {1: 'Apple', 'name': 'For', 3:
    'Apple'}
3.
4.  # mengakses elemen dengan key
5.  print("Mengakses sebuah elemen dengan
    key:")
6.  print(Dict['name'])
7.
8.  # mengakses sebuah elemen dengan key
9.  print("Mengakses sebuah elemen dengan
    key:")
10. print(Dict[1])
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Mengakses sebuah elemen dengan key:
For
Mengakses sebuah elemen dengan key:
Apple
```

Terdapat juga metode bernama get() yang juga akan membantu dalam mengakses elemen dari dictionary. Metode ini menerima key sebagai argumen dan mengembalikan value-nya.

File **dictionary.py**

```
1. # Membuat sebuah Dictionary
2. Dict = {1: 'Apple', 'name': 'For', 3:
   'Apple'}
3.
4. # mengakses sebuah elemen dengan metode
   get()
5. # method
6. print("Mengakses sebuah elemen dengan
   get:")
7. print(Dict.get(3))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Mengakses sebuah elemen dengan get:
Apple
```

3.5.4 Mengakses Elemen pada Nested Dictionary

Untuk mengakses nilai key apa pun dalam nested dictionary, gunakan sintaks pengindeksan [].

File **dictionary.py**

```
1. # Membuatsebuah dicionary
2. Dict = {'Dict1': {1: 'Apple'},
3.          'Dict2': {'Name': 'For'}}
4.
5. # Mengakses element dengan Key
6. print(Dict['Dict1'])
7. print(Dict['Dict1'][1])
8. print(Dict['Dict2']['Name'])
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
{1: 'Apple'}
Apple
For
```

3.5.5 Menghapus Elemen pada Dictionary

Item dictionary dapat dihapus dengan menggunakan kata kunci del seperti yang ada pada contoh kode berikut.

File **dictionary.py**

```
1. # Membuat Dictionary
2. Dict = {1: 'Apple', 'name': 'For', 3:
   'Apple'}
3.
4. print("Dictionary =")
5. print(Dict)
```

```
6. # Menghapus beberapa data Dictionary
7. del(Dict[1])
8. print("Data setelah penghapusan Dictionary
   =")
9. print(Dict)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Dictionary ={1: 'Apple', 'name': 'For',
3: 'Apple'}
Data         setelah         penghapusan
Dictionary={'name': 'For', 3: 'Apple'}
```

3.5.6 Metode pada Dictionary

Berikut ini adalah contoh metode yang bisa digunakan pada dictionary.

Tabel 2 Metode Dictionary

Method	**Deskripsi**
dic.clear()	Hapus semua elemen dari dictionary
dict.copy()	Mengembalikan salinan dictionary

dict.get(key, default = “None”)	Mengembalikan nilai kunci yang ditentukan
dict.items()	Mengembalikan daftar yang berisi tupel untuk setiap pasangan nilai kunci
dict.keys()	Mengembalikan daftar yang berisi kunci dictionary
dict.update(dict2)	Memperbarui dictionary dengan key-value pair yang ditentukan
dict.values()	Memperbarui dictionary dengan key-value pair yang ditentukan
pop()	Hapus elemen dengan kunci yang ditentukan
popItem()	Menghapus key-value pair yang terakhir dimasukkan
dict.setdefault(key,default= “None”)	atur kunci ke nilai default jika kunci tidak ditentukan dalam dictionary

dict.has_key(key)	mengembalikan nilai true jika dictionary berisi kunci yang ditentukan.
dict.get(key, default = “None”)	digunakan untuk mendapatkan nilai yang ditentukan untuk kunci yang diteruskan.

Berikut ini adalah contoh implementasi dari beberpa metode yang dapat digunakan pada dictionary.

File **dictionary.py**

```
1.  # demo for all dictionary methods
2.  dict1 = {1: "Python", 2: "Java", 3:
    "Ruby", 4: "Scala"}
3.
4.  # copy() method
5.  dict2 = dict1.copy()
6.  print(dict2)
7.
8.  # clear() method
9.  dict1.clear()
10. print(dict1)
11.
12. # get() method
13. print(dict2.get(1))
14.
15. # items() method
16. print(dict2.items())
```

```
17.
18. # keys() method
19. print(dict2.keys())
20.
21. # pop() method
22. dict2.pop(4)
23. print(dict2)
24.
25. # popitem() method
26. dict2.popitem()
27. print(dict2)
28.
29. # update() method
30. dict2.update({3: "Scala"})
31. print(dict2)
32.
33. # values() method
34. print(dict2.values())
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
{1: 'Python', 2: 'Java', 3: 'Ruby', 4:
'Scala'}
{}
Python
dict_items([(1, 'Python'), (2, 'Java'), (3,
'Ruby'), (4, 'Scala')])
dict_keys([1, 2, 3, 4])
{1: 'Python', 2: 'Java', 3: 'Ruby'}
{1: 'Python', 2: 'Java'}
```

```
{1: 'Python', 2: 'Java', 3: 'Scala'}
dict_values(['Python', 'Java', 'Scala'])
```

3.6 Set

Set atau Himpunan adalah kumpulan data unik. Artinya, elemen dari suatu himpunan tidak dapat digandakan.

Misalkan kita ingin menyimpan informasi tentang ID mahasiswa. Karena ID mahasiswa tidak dapat digandakan, kita dapat menggunakan satu set.

3.6.1 Membuat sebuah Set

Di Python, kita membuat set dengan menempatkan semua elemen di dalam kurung kurawal {}, dipisahkan dengan koma.

Satu set dapat memiliki sejumlah item dan mungkin dari jenis yang berbeda (integer, float, tuple, string, dll.). Tetapi satu set tidak dapat memiliki elemen yang dapat diubah seperti daftar, set, atau kamus sebagai elemennya. Kode berikut merupakan contoh dari set.

File **set.py**

```
1. # buat satu set tipe integer
2. mahasiswa_id = {112, 114, 116, 118, 115}
3. print('Mahasiswa ID:', mahasiswa_id)
```

```
4.
5.  # buat satu set tipe string
6.  huruf_vokal = {'a', 'e', 'i', 'o', 'u'}
7.  print('Huruf Vokal:', huruf_vokal)
8.
9.  # membuat satu set tipe data campuran
10. mixed_set = {'Hello', 101, -2, 'Bye'}
11. print('Kumpulan tipe data campuran:',
    mixed_set)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Mahasiswa ID: {112, 114, 115, 116, 118}
Huruf Vokal: {'o', 'u', 'i', 'e', 'a'}
Kumpulan tipe data campuran: {'Hello', 'Bye',
101, -2}
```

Dalam contoh di atas, kita telah membuat tipe set yang berbeda dengan menempatkan semua elemen di dalam kurung kurawal {}.

3.6.2 Membuat sebuah Set Kosong

Membuat set kosong agak rumit. Kurung kurawal kosong {} akan membuat kamus kosong di Python. Untuk membuat set tanpa elemen, kita menggunakan fungsi set() tanpa argumen. Misalnya.

File **set.py**

```
1.  # membuat set kosong
2.  empty_set = set()
3.
4.  # membuat dictionary kosong
5.  empty_dictionary = { }
6.
7.  # periksa tipe data dari empty_set
8.  print('Jenis data dari empty_set:',
    type(empty_set))
9.
10. # periksa tipe data dari dictionary_set
11. print('Jenis data dari empty_dictionary',
    type(empty_dictionary))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Jenis data dari empty_set: <class 'set'>
Jenis data dari empty_dictionary <class
'dict'>
```

Berdasarkan kode di atas empty_set adalah set kosong yang dibuat menggunakan set(), empty_dictionary adalah dictionary kosong yang dibuat menggunakan {}. Selanjutnya kita juga telah menggunakan fungsi type() untuk mengetahui jenis kelas dari empty_set dan empty_dictionary.

3.6.3 Duplikasi item dalam Set

Jika kita mencoba menyertakan item duplikat dalam satu set, maka tidak ada item duplikat di set karena set tidak dapat berisi duplikat. Berikut ini adalah contohnya.

File **set.py**

```
1. numbers = {2, 4, 6, 6, 2, 8}
2. print(numbers)   # {8, 2, 4, 6}
```

3.6.5 Menambahkan item pada Set

Set dapat berubah. Namun, karena tidak diurutkan, pengindeksan tidak ada artinya. Kita tidak dapat mengakses atau mengubah elemen dari Set menggunakan pengindeksan atau pemotongan (slice).

Di Python, kita bisa menggunakan metode add() untuk menambahkan item ke set. Misalnya dapat dilihat pada kode berikut.

File **set.py**

```
1. numbers = {21, 34, 54, 12}
2.
3. print('Initial Set:',numbers)
4.
5. # menggunakan add() method
6. numbers.add(32)
7.
8. print('Updated Set:', numbers)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Initial Set: {34, 12, 21, 54}
Updated Set: {32, 34, 12, 21, 54}
```

Dalam contoh di atas, kita telah membuat satu set dengan nama numbers. Perhatikan baris ini numbers.add(32). Di sini, add() menambahkan 32 ke set kita.

3.6.6 Mengupdate item pada Set

Metode update() digunakan untuk memperbarui set dengan item jenis koleksi lainnya (list, tupel, set, dll). Misalnya dapat dilihat pada kode berikut.

File **set.py**

```
1. companies = {'UHB', 'UAD'}
2. tech_companies = ['apple', 'google',
   'apple']
3.
4. companies.update(tech_companies)
5.
6. print(companies)
7.
8. # Output: {'google', 'apple', 'Lacoste',
   'Ralph Lauren'}
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
{'UAD', 'google', 'apple', 'UHB'}
```

Di sini, semua elemen unik tech_companies ditambahkan ke set companies.

3.6.6 Remove item pada Set

Kami menggunakan metode discard() untuk menghapus elemen yang ditentukan dari set. Misalnya dapat dilihat pada kode berikut.

File **set.py**

```
1. languages = {'Swift', 'Java', 'Python'}
2.
3. print('Set Awal:',languages)
4.
5. # hapus 'Java' dari sebuah set
6. removedValue = languages.discard('Java')
7.
8. print('Set setelah menggunakan remove():',
   languages)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Set Awal: {'Python', 'Swift', 'Java'}
Set setelah menggunakan remove(): {'Python',
'Swift'}
```

Di sini, kami telah menggunakan metode dicard() untuk menghapus 'Java' dari set languages.

3.6.7 Fungsi bawaan Set

Fungsi bawaan seperti all(), any(), enumerate(), len(), max(), min(), sorted(), sum() dll. biasanya digunakan dengan set untuk melakukan tugas yang berbeda.

Function	Deskripsi
all()	Mengembalikan True jika semua elemen himpunan benar (atau jika himpunan kosong).
any()	Mengembalikan True jika ada elemen dari himpunan yang benar. Jika set kosong, kembalikan False.
enumerate()	Mengembalikan objek enumerasi. Ini berisi indeks dan nilai untuk semua item dari set sebagai pasangan.
len()	Mengembalikan panjang (jumlah item) di set.
max()	Mengembalikan item terbesar di set.

min()	Mengembalikan item terkecil di set.
sorted()	Mengembalikan list terurut baru dari elemen di set (tidak mengurutkan set itu sendiri).
sum()	Mengembalikan jumlah semua elemen dalam set.

Kita dapat menggunakan perulangan for untuk mengulangi elemen pada set.

File **set.py**

```
1. fruits = {"Apple", "Peach", "Mango"}
2.
3. # for loop untuk mengakses setiap fruits
4. for fruit in fruits:
5.     print(fruit)
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Mango
Peach
Apple
```

Kita dapat menggunakan metode len() untuk menemukan jumlah elemen yang ada dalam sebuah Set.

File **set.py**

```
1. even_numbers = {2,4,6,8}
2. print('Set:',even_numbers)
3.
4. # mencari panjang atau jumlah elements
5. print('Total Elements:',
   len(even_numbers))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Set: {8, 2, 4, 6}
Total Elements: 4
```

3.6.8 Operasi pada Set

Python Set menyediakan metode bawaan yang berbeda untuk melakukan operasi himpunan matematika seperti union, intersection, subtraction, dan symmetric difference.

1. Union dari Dua Sets

   Gabungan dua himpunan A dan B mencakup semua anggota himpunan A dan B. Kita menggunakan | operator atau metode union()

untuk melakukan operasi gabungan yang ditetapkan.

File **set.py**

```
1. # first set
2. A = {1, 3, 5}
3.
4. # second set
5. B = {0, 2, 4}
6.
7. # perform union operation using |
8. print('Union using |:', A | B)
9.
10. # perform union operation using
    union()
11. print('Union using union():',
    A.union(B))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Union using |: {0, 1, 2, 3, 4, 5}
Union using union(): {0, 1, 2, 3, 4, 5}
```

A | B dan union() setara dengan operasi set

A ∪ B.

2. Set Intersection

Irisan dua himpunan A dan B termasuk elemen persekutuan antara himpunan A dan B. Di Python, kita dapat menggunakan operator '&' atau metode persimpangan () untuk melakukan operasi persimpangan yang ditetapkan.

File **set.py**

```
1.  # first set
2.  A = {1, 3, 5}
3.
4.  # second set
5.  B = {1, 2, 3}
6.
7.  # perform intersection operation using &
8.  print('Intersection using &:', A & B)
9.
10. # perform intersection operation using
    intersection()
11. print('Intersection using
    intersection():', A.intersection(B))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Intersection using &: {1, 3}
Intersection using intersection():
{1, 3}
```

3. Difference between Two Sets

   Selisih antara dua himpunan A dan B termasuk anggota himpunan A yang tidak ada pada himpunan B. Kita menggunakan operator "-" atau metode difference() untuk melakukan perbedaan antara dua set.

File **set.py**

```
1.  # first set
2.  A = {2, 3, 5}
3.
4.  # second set
5.  B = {1, 2, 6}
6.
7.  # perform difference operation using &
8.  print('Difference using &:', A - B)
9.
10. # perform difference operation using
    difference()
11. print('Difference using difference():',
    A.difference(B))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Difference using &: {3, 5}
Difference using difference(): {3, 5}
```

4. Set Symmetric Difference

   Selisih simetris antara dua himpunan A dan B mencakup semua elemen A dan B tanpa elemen persekutuan. Dalam Python, kita dapat menggunakan operator ^ atau metode symmetric_difference() untuk melakukan perbedaan simetris antara dua set.

File **set.py**

```
1.  # first set
2.  A = {2, 3, 5}
3.
4.  # second set
5.  B = {1, 2, 6}
6.
7.  # perform difference operation using &
8.  print('using ^:', A ^ B)
9.
10. # using symmetric_difference()
11. print('using symmetric_difference():',
    A.symmetric_difference(B))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
using ^: {1, 3, 5, 6}
using symmetric_difference(): {1, 3,
5, 6}
```

3.6.9 Periksa apakah dua set sama

Kita dapat menggunakan operator == untuk memeriksa apakah dua set sama atau tidak.

File **set.py**

```
1.  # first set
2.  A = {1, 3, 5}
3.
4.  # second set
5.  B = {3, 5, 1}
6.
7.  # perform difference operation using &
8.  if A == B:
9.      print('Set A and Set B are equal')
10. else:
11.     print('Set A and Set B are not
    equal')print('using
    symmetric_difference():',
    A.symmetric_difference(B))
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Set A and Set B are equal
```

BAB IV
Stack & Queue

Pada bab sebelumnya kita telah belajar tentang dasar penggunaan *Array* yang secara umum menggunakan *array list*. Modul ini akan membahas konsep stack (tumpukan) dan queue (antrian) dalam struktur data dengan bahasa python .

Hampir setiap hari kita bisa melihat objek yang bertumpuk seperti buku atau tumpukan baju. Begitu juga dengan antrian, seperti halnya kita sedang menunggu giliran untuk mendapatkan secangkir kopi atau menu makanan di sebuah café. Dalam ilmu komputer kedua hal tersebut akan sering kita jumpai.

Stack menggunakan prinsip LIFO (Last In First Out). Cara kerjanya seperti ada sebuah tumpukan baju di dalam lemari dan baju yang paling atas (Last In) terlebih dahulu kita ambil dan kenakan (First Out). Penggunaan stack bisa dilakukan di List. Kita akan mencoba menerapkan beberapa latihan untuk memahami bagaimana stack bekerja.

Queue atau antrian menggunakan prinsip FIFO (First In First Out). Cara kerjanya seperti halnya anda pergi mengantri di bioskop, siapa yang mengantri paling

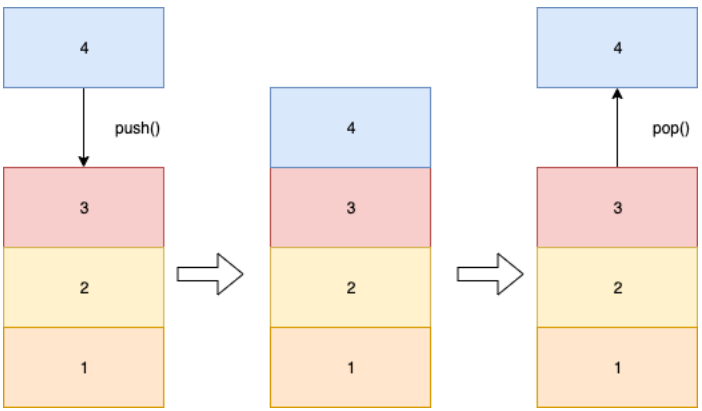
depan maka dia yang mendapatkan tiket terlebih dahulu.

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *terminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi
Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

4.1 Stack

Implementasi stack dapat dilakukan dengan menggunakan List pada python, kita dapat memanfaatkan fungsi push () dan pop (). Ilustrasi stack dapat dilihat pada gambar 4.1



Gambar 2 Ilustrasi Stack/LIFO

Berdasarkan gambar 4.1 dalam sebuah tumpukan awal terdapat tiga element yaitu angka 1, 2 dan 3 kemudian dengan operasi push () kita tambahkan element 4 sehingga tumpukan menjadi 1,2,3 dan empat. Selanjutnya kita menjalankan konsep LIFO yang mana element terakhir yang dimasukan digunakan atau dikeluarkan terlebih dahulu dengan operasi pop () jadi karena 4 adalah element terakhir maka element tersebut yang dikeluarkan.

Kode berikut merupakan contoh penggunaan konsep stack pada bahasa pemrograman python dengan memanfaatkan list.

Pada code **stack-list.py** anda akan melihat penggunaan *list*.

File **stack-list.py**

```
1.  #list kata berisi tumpukan kosong
2.  kata = []
3.
4.  # tambahkan beberapa elemen dengan append()
5.  kata.append('n') #0
6.  kata.append('a') #1
7.  kata.append('m') #2
8.  kata.append('a') #3
9.
10. # Kita jalankan pop()
11. # huruf a di dapatkan dan di keluarkan
12. kata_terakhir = kata.pop()
```

```
13. print(kata_terakhir)
14.
15. # Kita Jalankan pop() kembali
16. # huruf terakhir sekarang adalah m
17. kata_terakhir = kata.pop()
18. print(kata_terakhir)
19.
20. # cetak isi tumpukan yang masih tersedia
21. print(kata) #output ['n', 'a']
```

Python memiliki library *deque* yang bisa diimplementasikan juga untuk pembuatan stack. Kode berikut menjelaskan stack dengan deque.

File **stack-deque.py**

```
1.  from collections import deque
2.
3.  # menyiapkan list dengan deque
4.  numbers = deque()
5.
6.  # tambahkan elemen baru ke dalam numbers
7.  numbers.append(99)
8.  numbers.append(15)
9.  numbers.append(82)
10. numbers.append(50)
11. numbers.append(47)
12.
13. # Jalankan pop() untuk mengeluarkan elemen
    yang terakhir dimasukan
14. last_item = numbers.pop()
15. print(last_item) # 47
```

```
16. print(numbers) # deque([99, 15, 82, 50])
```

Lebih jauh lagi kita bisa mengimplementasikan Stack dengan membuat kelas seperti pada konsep OOP (Object Oriented Programming) seperti berikut.

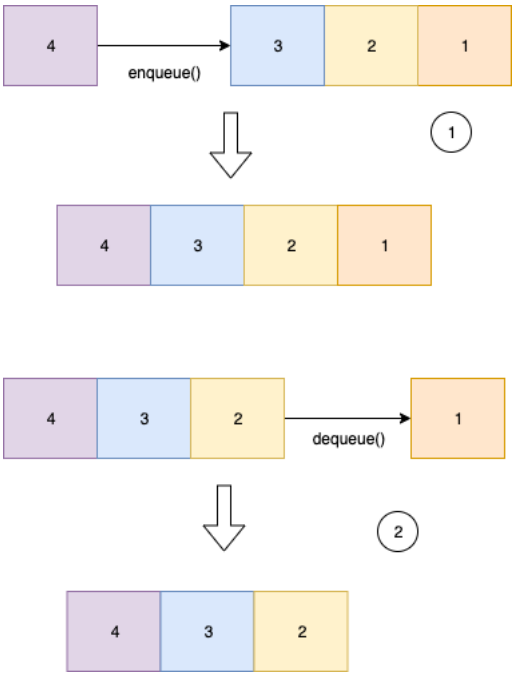
File **stack-class.py**

```
1.  #buat kelas dengan nama Stack
2.  class Stack:
3.
4.      #fungsi yang pertama kali dijalankan
5.      #fungsi pernyiapan class dimulai dengan
    list kosong []
6.      def __init__(self):
7.          self.stack = []
8.
9.      #fungsi untuk menghapus elemen terakhir
10.     def pop(self):
11.         if len(self.stack) < 1:
12.             return None
13.         return self.stack.pop()
14.
15.     #fungsi untuk menambahkan elemen
16.     def push(self, item):
17.         self.stack.append(item)
18.
19.     #fungsi untuk melihat ukuran dari Stack
20.     def size(self):
21.         return len(self.stack)
22.
```

```
#buat objek baru dengan nama buku, dia mewarisi semua sifat dari class Stack
buku = Stack()

# tambahkan elemen dengan push
buku.push('Algoritma')
# tambahkan buku baru dengan push
buku.push('Pemrograman Web')
# Keluarkan elemen terakhir dari stack buku
buku.pop()
# tambahkan buku baru
buku.push('Automata')

#lihat data buku
print(buku.stack)
```

4.2 Queue

Implementasi *Queue* juga dapat dilakukan pada *list* python. Pada queue kita akan mengenal teknik *enqueue* yaitu cara untuk menambahkan element pada posisi terakhir antrian. Sedangkan *dequeue* adalah memilih dan mengeluarkan element pada posisi awal. Ilustrasi dari queue dapat dilihat pada Gambar 4.2.



Gambar 3 Queue

Sebagai contoh kita akan menggunakan *list* untuk operasi *queue*. Kita dapat menggunakan fungsi yang sama untuk mengimplementasikan Queue. Fungsi pop secara opsional mengambil indeks item yang ingin kita ambil sebagai argumen.

File queue.py

```
1. kata = []
2.
3. # tambahkan beberapa elemen dengan append()
4. kata.append('n') #0
5. kata.append('a') #1
6. kata.append('m') #2
7. kata.append('a') #3
```

```
8.
9.  # Sekarang dequeue elemen pertama
10. # 'n' di hapus
11. first_item = kata.pop(0)
12. print(first_item)
13.
14. # dequeue kembali
15. # 'a' dihapus
16. first_item = kata.pop(0)
17. print(first_item)
18.
19. # 'n' and 'a' terhapus
20. print(kata) # ['m', 'a']
```

Python memiliki library *deque* yang bisa diimplementasikan juga untuk pembuatan stack. Kode berikut menjelaskan queue dengan deque.

File dequeue.py

```
1.  from collections import deque
2.
3.  # siapkan dequeue list kosong
4.  numbers = deque()
5.
6.  # Tambahkan beberapa elemen dengan append
7.  numbers.append(99)
8.  numbers.append(15)
9.  numbers.append(82)
10. numbers.append(50)
11. numbers.append(47)
12.
13.
```

```
14. # lakukan queue untuk menghapus elemen
    pertama
15. # gunakan fungsi popleft()
16. first_item = numbers.popleft()
17. print(first_item) # 99
18. print(numbers) # deque([15, 82, 50,47])
```

Lebih jauh lagi kita bisa mengimplementasikan Queue dengan membuat kelas seperti pada konsep OOP (Object Oriented Programming) seperti berikut.

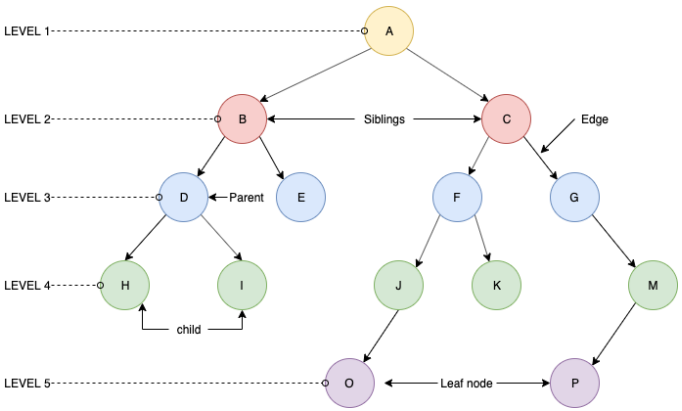
File **dequeue-class.py**

```
1.  #buat kelas baru
2.  class Queue:
3.
4.      #fungsi yang pertama kali dijalankan
5.      #siapkan dengan list kosong
6.      def __init__(self):
7.          self.queue = []
8.
9.      #tambahkan elemen ke antria
10.     def enqueue(self, item):
11.         self.queue.append(item)
12.
13.     #proses antrian yang pertama
14.     def dequeue(self):
15.         #jika panjang antrian kurang dari
    satu maka abaikan
16.         if len(self.queue) < 1:
17.             return None
18.         #jika lebih dari satu maka hapus
    elemen pertama
```

```
        return self.queue.pop(0)

    #fungsi cek panjang antrian
    def size(self):
        return len(self.queue)

#buat objek baru dari class Queue
#mewarisi semua sifat class induknya
antrian = Queue()

# Tambahkan antrian
antrian.enqueue('Salman')
antrian.enqueue('Hari')
antrian.enqueue('Keenar')
antrian.enqueue('Zira')
antrian.enqueue('Noval')

# lakukan dequeue
# 'hapus Salman' => posisi 0
prosesantrian = antrian.dequeue()

# dequeue kembali
# 'hapus Hari' => posisi 0
prosesantrian = antrian.dequeue()

# dequeue kembali
# 'hapus Keenar' => posisi 0
prosesantrian = antrian.dequeue()

print(antrian.queue) # ['Zira','Noval']
```

BAB V
Tree

Tree adalah struktur data non-linear yang mewakili node yang dihubungkan oleh tepi. Setiap pohon terdiri dari simpul akar *(root node)* sebagai simpul Induk *(parent)*, dan simpul kiri *(left node)* dan simpul kanan *(right node)* sebagai simpul Anak *(child)*. Tree dapat diilustrasikan pada Gambar 5



Gambar 4 Tree

Kita memiliki 5 level dalam Tree, level 1 dimulai dengan akar tree (root). Node D merupakan parent dari node H dan I, sehingga H & I merupakan node child. Edge adalah garis pehubung antar node. Node yang berjejer merupakan siblings. Node dengan

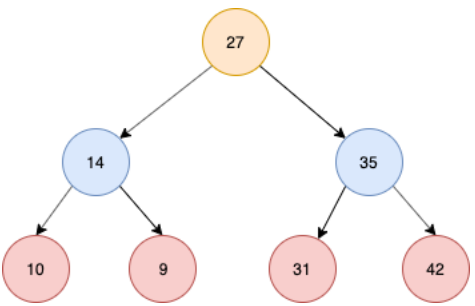
posisi paling ujung disebut juga dengan node leaf (daun) yang terdapat pada lebel 5.

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *terminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi
Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

5.1 Binary Tree

Pohon yang elemennya paling banyak memiliki dua anak disebut pohon biner. Setiap element dalam pohon biner hanya dapat memiliki dua anak. Anak kiri simpul harus memiliki nilai lebih kecil dari nilai induknya, dan anak kanan simpul harus memiliki nilai lebih besar dari nilai induknya.



Gambar 5 Binary Tree

5.2 Implementasi Binary Tree

5.2.1 Pembuatan Node Root

Kode berikut merupakan contoh penggunaan konsep binary tree pada bahasa pemrograman python. Kelas Node dibuat untuk menciptakan *node root*.

Pada code **node-tree.py**.

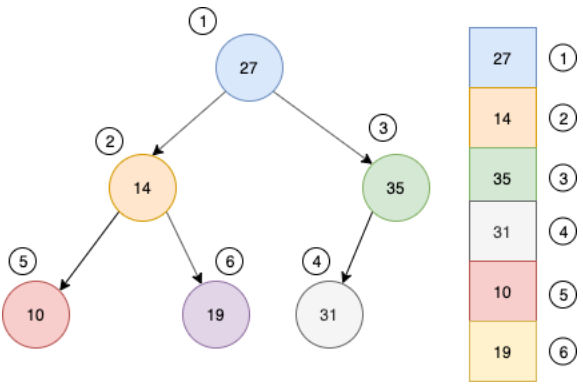
File **binary-tree.py**

```
1.  # node class
2.  class Node:
3.
4.      def __init__(self, data):
5.          # left child
6.          self.left = None
7.          # right child
8.          self.right = None
9.          # node's value
10.         self.data = data
11.
12.     # print function
13.     def PrintTree(self):
14.         print(self.data)
15.
16. root = Node(27)
17.
18. root.PrintTree() #ouput 27
```

5.2.2 Menyisipkan node

Metode insert () membandingkan kondisi nilai simpul ke simpul induk dan memutuskan apakah akan menambahkannya sebagai simpul kiri atau simpul kanan. Jika simpul lebih besar dari simpul induk, itu dimasukkan sebagai simpul kanan; jika tidak, itu dimasukkan ke kiri. Akhirnya, metode PrintTree digunakan untuk mencetak tree.

Sebagai contoh pada sebelumnya kita telah menambahkan node root dengan nilai 45, kemudian root tersebut kita tambahkan nilai 14, 35, 31, 10 dan 19. Gambar tree yang dihasilkan adalah sebagai berikut.



Gambar 6 Urutan Tree

Implementasi tree tersebut dapat dilihat pada kode berikut ini.

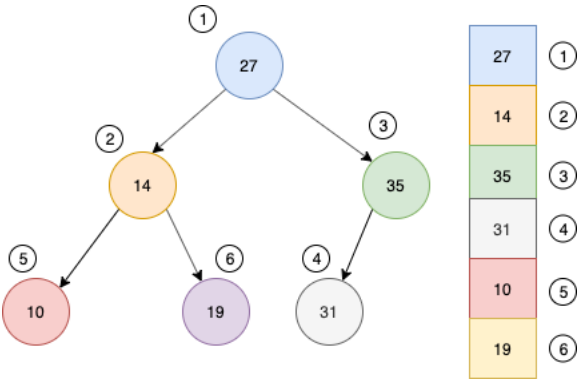
File **tree-insert.py**

```
1.  class Node:
2.
3.      def __init__(self, data):
4.
5.          self.left = None
6.          self.right = None
7.          self.data = data
8.
9.      def insert(self, data):
10.    # Bandingkan nilainya dengan parent
    dimulai dari root
11.         if self.data:
12.             # Jika nilai lebih kecil maka
    node ke kiri
13.             if data < self.data:
14.                 if self.left is None:
15.                     self.left = Node(data)
16.                 else:
17.                     self.left.insert(data)
18.             # Jika nilai lebih besar maka
    node ke kanan
19.             elif data > self.data:
20.                 if self.right is None:
21.                     self.right =
    Node(data)
22.                 else:
23.
    self.right.insert(data)
24.         else:
```

```
            self.data = data

# Print the tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()

# Use the insert method to add nodes
root = Node(27)
root.insert(14)
root.insert(35)
root.insert(31)
root.insert(10)
root.insert(19)


root.PrintTree()
```

5.2.3 Pencarian Sebuah Node

Kita juga bisa mencari nilai dari *tree* yang telah dibuat. Saat mencari nilai di *tree*, kita perlu melintasi node dari kiri ke kanan dan dengan parentnya. Kita harus terlebih dahulu membandingkan nilai yang dicari sebagai contoh pada tree yang sudah kita buat sebelumnya pada Gambar 5.4. Ketika kita misalnya mencari nilai 19 maka langkah kerjanya adalah sebagai berikut:



Gambar 7 Langkah Kerja Pencarian Node 19

1. Bandingkan 19 dengan dengan nilai root yaitu 27
2. Cari nilainya apakah lebih besar aatau lebih kecil, karena 19 lebih kecil dari 27 maka pencarian berarti ke node sebelah kiri.
3. Pada posisi sebelah kiri kemudian 19 dibandingkan dengan nilai 14 dan karena 19 lebih besar dari 14 maka pindah ke node kiri
4. Node 19 ditemukan.

Metode pencarian node yang dapat dibuat pada *class node* adalah sebagai berikut.

File **search-node.py**

```
1.  # metode untuk mencari sebuah value
2.      def findval(self, lkpval):
3.          #cari di node kiri dahulu
4.          if lkpval < self.data:
```

```
            #jika di kiri kosong
            if self.left is None:
                return str(lkpval)+" Tidak Ditemukan"
            return self.left.findval(lkpval)
        #jika di kiri tidak ada cari ke node kanan
        elif lkpval > self.data:
            #jika di kanan kosong
            if self.right is None:
                return str(lkpval)+" Tidak Ditemukan"
            return self.right.findval(lkpval)
        # node ditemukan
        else:
            return str(self.data) + " Ditemukan"
```

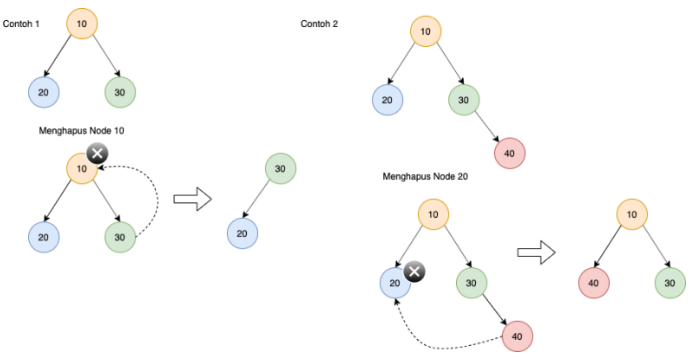
Untuk menggunakan method tersebut kita dapat mengimplementasikannya diluar kelas Node setelah anda menambahkan node sebagai berikut.

File **search-node.py**

```
print(root.findval(13)) #tidak ditemukan
print(root.findval(14)) #ditemukan
```

5.2.4 Menghapus Sebuah Node

Diberikan ***binary tree***, hapus ***node*** darinya dengan memastikan bahwa ***tree*** menyusut dari bawah (yaitu ***node*** yang dihapus diganti dengan ***node*** paling bawah dan paling kanan). Di sini kita tidak memiliki urutan antar element, jadi kita ganti dengan element terakhir.



Gambar 8 Menghapus Node

Algoritma penghapusan dalam binary tree adalah sebagai berikut:

1. Mulai dari root, temukan simpul terdalam dan paling kanan di *binary tree* dan *node* yang ingin kita hapus.
2. Ganti data node paling kanan terdalam dengan node yang akan dihapus.
3. Kemudian hapus simpul paling kanan terdalam.

Implementasi dari bahasa pemrograman python adalah sebagai berikut.

File **delete-binary-tree-node.py**

```
1.      # berfungsi untuk menghapus node
    masih menjadi metode
2.      #Kelas Node
3.      # metode untuk menghapus node tertentu
    dengan node terdalam
4.      def deleteDeepest(self,d_node):
5.          q = []
6.          q.append(self)
7.          while(len(q)):
8.              temp = q.pop(0)
9.              if temp is d_node:
10.                 temp = None
11.                 return
12.             if temp.right:
13.                 if temp.right is d_node:
14.                     temp.right = None
15.                     return
16.                 else:
17.                     q.append(temp.right)
18.             if temp.left:
19.                 if temp.left is d_node:
20.                     temp.left = None
21.                     return
22.                 else:
23.                     q.append(temp.left)
24.
25.     # Fungsi untuk menghapus node
26.     def deletion(self, key):
27.         if self == None :
28.             return None
```

```
        if self.left == None and
    self.right == None:
            if self.data == key :
                return None
            else :
                return self
        key_node = None
        q = []
        q.append(self)
        while(len(q)):
            temp = q.pop(0)
            if temp.data == key:
                key_node = temp
            if temp.left:
                q.append(temp.left)
            if temp.right:
                q.append(temp.right)
        if key_node :
            x = temp.data
            self.deleteDeepest(temp)
            key_node.data = x
        return self
```

Untuk melihat dan menjalankan fungsi tersebut bisa dilakukan dengan cara sebagai berikut.

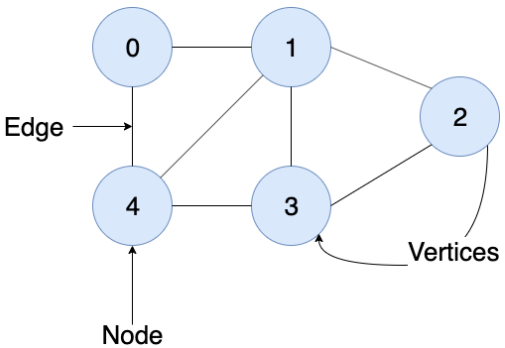
File **delete-binary-tree-node.py**

```
root = Node(27)
root.insert(14)
root.insert(35)
root.insert(31)
```

```
5. root.insert(10)
6. root.insert(19)
7.
8. print('Sebelum di hapus')
9. root.PrintTree()
10.
11. root.deletion(35)
12.
13. print('Setelah di hapus')
14. root.PrintTree()
```

BAB VI
Graph

Selama kita menggunakan media sosial seperti facebook, kita tentunya terhubung dengan orang lain dalam jalinan pertemanan. Interaksi sejenis ini membentuk suatu tautan yang disebut *Graph*. Disini kita dan orang lain merupakan *node* (simpul) dalam *graph* dan permintaan pertemanan akan berubah menjadi *edge* (tepi) antara kita dan orang lain. Selain terhubung dalam bentuk pertemanan kita juga bisa terhubung dengan node lainnya seperti group, event dan sebagainya. Dengan demikian, keseluruhan jaringan facebook merupakan kumpulan besar simpul (*vertices*) dan *edges*. Graph dapat diilustrasikan pada Gambar 6



Gambar 9 Graph

Struktur data *graph* adalah semua tentang *node* dan *edge*. *Graph* adalah struktur data (V, E) yang terdiri dari:

- Kumpulan *vertices* (V)
- Kumpulan *edge* (E), diwakili oleh pasangan *vertices* (u, v).

6.1 Dasar Graph

Perhatikan himpunan *vertices* dan *edge* di bawah ini, kami memiliki 4 *vertices* dan 4 edge, *node* / *vertex* 0 terhubung langsung dengan semua *vertices* dan 1 terhubung dengan 2, tetapi tidak ada edge langsung antara (1, 3) dan (2, 3).

V = {0, 1, 2, 3}

E = {(0,1), (0,2), (0,3), (1,2)}

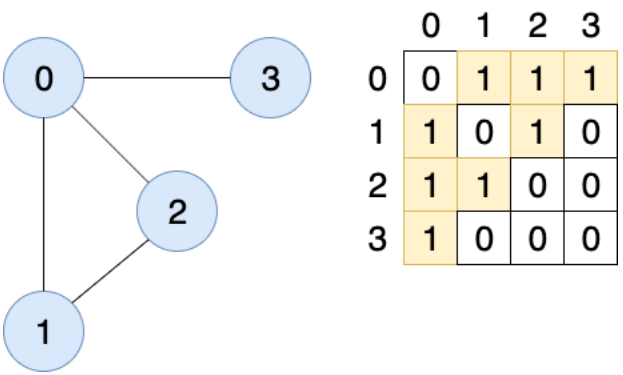
G = {V, E}

Adjacency: Suatu *node/vertex* dikatakan bertetangga dengan *nod/vertexe* lain jika ada *edge* yang menghubungkannya.

Path: Urutan *edge* yang memungkinkan Anda berpindah dari *vertex* A ke *vertex* B disebut *path*. 0–1, 1–2 dan 0–2 adalah lintasan dari *vertex* 0 ke *vertex* 2.

Connected Graph: *Graph* yang selalu ada lintasan dari suatu *vertex* ke *vertex* lainnya. Jalur di sini mengacu pada jalur langsung atau jalur tidak langsung antara *vertices*.

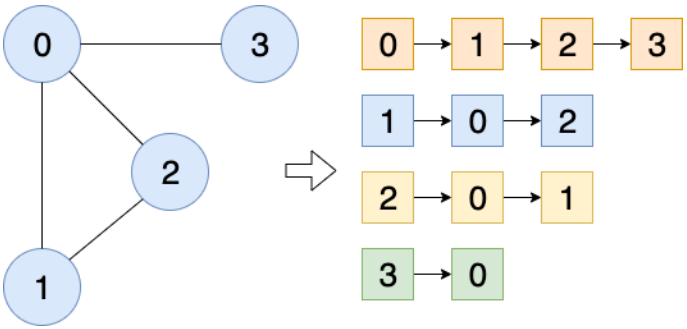
Representasi Graph



Gambar 10 Adjacency Matrix

Karena merupakan *graph* tak berarah, untuk sisi (0,2), kita juga perlu menandai sisi (2,0); membuat matriks ketetanggaan simetris terhadap diagonal.

Pencarian edge (memeriksa apakah ada edge antara node A dan node B) sangat cepat dalam representasi matriks ketetanggaan tetapi kita harus memesan ruang untuk setiap kemungkinan tautan antara semua node (V x V), sehingga membutuhkan lebih banyak ruang.



Gambar 11 Alur Pencarian Edge Adjacency Matrix

Berdasarkan gambar 6.2 node 0 mencari node yang terhubung dengan edge terdekat, jadi node 0 menuju node 1 lalu node 2 dan node 3. Node 1 ke node 0 dan node 2. Node 2 ke node 0 dan node 1. Node 3 hanya terhubung dengan node 0.

List adjacency efisien dalam hal penyimpanan karena kita hanya perlu menyimpan nilai untuk edge. Untuk *graph* dengan jutaan *node*, ini bisa berarti banyak ruang yang dihemat.

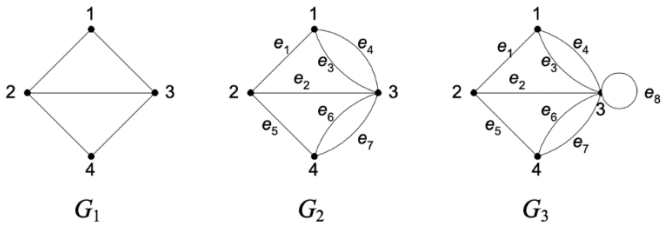
Graph yang merepresentasikan jembatan Königsberg:

- Simpul (vertex) menyatakan daratan
- Sisi (edge) menyatakan jembatan

Graf $G = (V, E)$, yang dalam hal ini:

- V = himpunan tidak-kosong dari simpul-simpul $(vertices) = \{ v1, v2, \ldots, vn \}$
- E = himpunan sisi (edges) yang menghubungkan sepasang $simpul =$

$\{e1, e2, \ldots, en\}$



Gambar 12 Graf

Contoh 1. Pada Gambar 6.3,

G1 adalah graf dengan

$V = \{1, 2, 3, 4\}$

$E = \{(1,2), (1,3), (2,3), (2,4), (3,4)\}$

G2 adalah graf dengan

$V = \{1, 2, 3, 4\}$

E

$= \{(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4)\}$

$= \{e1, e2, e3, e4, e5, e6, e7\}$

G3 adalah graf dengan

$V = \{1, 2, 3, 4\}$

E

$= \{(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4), (3,3)\}$

$= \{e1, e2, e3, e4, e5, e6, e7, e8\}$

- Pada G2, sisi e3 = (1, 3) dan sisi e4 = (1, 3) dinamakan sisiganda (multiple edges atau paralel edges) karena kedua sisiini

menghubungi dua buah simpul yang sama, yaitu simpul 1dan simpul 3.

- Pada G3, sisi e8 = (3, 3) dinamakan gelang atau kalang (loop)
  karena ia berawal dan berakhir pada simpul yang sama.

6.2 Operasi Graph

Ada beberapa operasi dasar *graph* yang bisa dijalankan antara lain adalah:

- Cari (Search)
- Lintasan Grafik (Graph Traversing)
- Insersi (Insert)
- Menemukan jalur dari satu *simpul* ke *simpul* lainnya (Find)

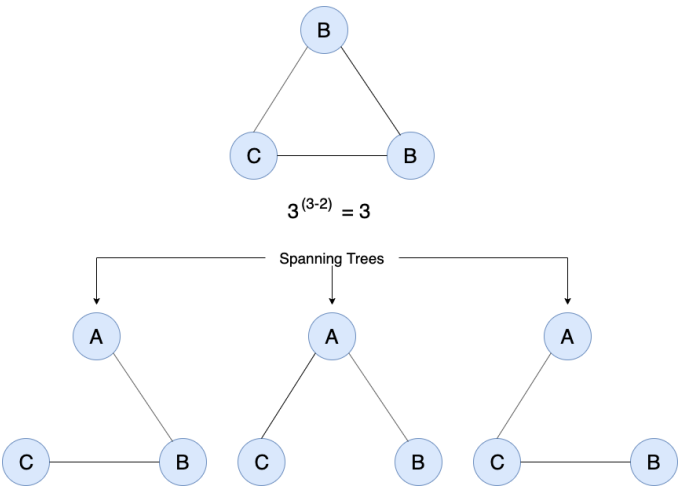
Struktur *graph* yang banyak digunakan adalah *Spanning Tree* dan variannya *Minimum Spanning Tree*.

Spanning Tree: Ini adalah *sub-graph* dari *graph* terhubung tak berarah, yang mencakup semua *node* yang terhubung baik secara langsung maupun tidak langsung dengan jumlah sisi paling sedikit. Jika sebuah *node* dibiarkan sendiri maka *sub-graph* tersebut bukan merupakan *spanning tree* dan *edge-*

nya mungkin atau mungkin tidak memiliki bobot yang diberikan padanya.

Jika suatu *graph* lengkap terdiri dari n *node*, maka $n\hat{}(n-2)$ *sub-graph* atau *spanning tree* mungkin terjadi.

Jika kita memiliki n = 3, jumlah *spanning tree* maksimum yang mungkin adalah $3^{(3-2)} = 16$. Jadi hasilnya adalah, $3^{(2-1)}$ *spanning tree* dapat dibentuk dari *graph* lengkap dengan 4 simpul. Untuk mempermudah pemahaman silahkan lihat ilustrasi berikut.



Gambar 13 Spanning Tree

Minimum Spanning Tree (MST) adalah *tree* yang jumlah bobot (*edge*) sisi-sisinya seminimal mungkin. MST adalah graph berbobot.

Untuk mencari *spanning tree minimum* dalam suatu *graph*, kita dapat menggunakan dua algoritma:

- Algoritma Prim
- Algoritma Kruskal

6.3 Matriks Adjacency dan Implementasinya

Matriks *Adjacency* A, adalah cara merepresentasikan graph $G = (V, E)$ sebagai matriks *boolean*.

Representasi Matriks: VxV, V adalah jumlah *vertices* dan setiap entri dalam matriks diwakili oleh A_ij, nilainya adalah 1 atau 0 tergantung pada apakah ada *edge* dari *vertex* i ke *vertex* j.

Dalam kasus graph tak berarah, matriksnya simetris terhadap diagonal karena setiap edge (i, j), ada juga edge (j, i), graph tak berarah adalah dua arah.

Mengapa Matriks *Adjacency*?

Operasi seperti menambahkan *edge*, menghilangkan *edge*, memeriksa apakah jalur dari *vertex* i ke *vertex* j ada atau tidak efisien waktu dengan waktu konstan.

Bahkan jika terdapat banyak vertices atau edge, kita dapat menggunakan matriks *adjacency*. Bahkan untuk *matriks sparse adjacency*, kita dapat menggunakan struktur data yang relevan dengannya.

Kekurangan Matriks *Adjacency*

Matriks V x V adalah memori intensif dan biasanya sebagian besar matriks tidak sepenuhnya terhubung, membuat *list Adjacency* pilihan yang lebih disukai untuk sebagian besar tugas. Sementara operasi dasar mudah, operasi seperti ***inEdges*** dan ***outEdges*** mahal saat menggunakan representasi matriks *Adjacency*.

Anda dapat mencoba mengeksekusi code tersebut dengan menjalankan perintah pada *terminal console* atau *command prompt* dengan mengetikan **"python3 namafile.py".**

Informasi
Kode selengkapnya dapat diakses di *https://github.com/ipung-uhb/data-structures*

File **graph.py**

```
1.  class Graph:
2.      def __init__(self, size):
3.          self.adjacency_matrix = []
4.          for i in range(size):
```

```
5.
    self.adjacency_matrix.append([0 for i in
    range(size)])
6.          self.size = size
7.
8.      def add_edge(self, v1, v2):
9.          if v1 == v2:
10.             print("HUbungkan dengan
    vertex/node yang sama")
11.         self.adjacency_matrix[v1][v2] = 1
12.         self.adjacency_matrix[v2][v1] = 1
13.
14.     def remove_edge(self, v1, v2):
15.         if self.adjacency_matrix[v1][v2]
    == 0:
16.             print("Tidak ada edge antara
    %d dan %d" % (v1, v2))
17.             return
18.         self.adjacency_matrix[v1][v2] = 0
19.         self.adjacency_matrix[v2][v1] = 0
20.
21.     def __len__(self):
22.         return self.size
23.
24.     def print_matrix(self):
25.         for row in self.adjacency_matrix:
26.             print(row)
27.
28. g = Graph(5)
29. g.add_edge(0, 1)
30. g.add_edge(0, 2)
31. g.add_edge(1, 2)
32. g.add_edge(2, 0)
33. g.add_edge(2, 3)
```

```
34.
35. g.print_matrix()
36.
37. #output
38. '''[0, 1, 1, 0, 0]
39. [1, 0, 1, 0, 0]
40. [1, 1, 0, 1, 0]
41. [0, 0, 1, 0, 0]
42. [0, 0, 0, 0, 0]'''
```

List Adjaency

Ini mewakili struktur data *graph* menggunakan array daftar tertaut (*linked list*). Setiap indeks dalam *array* mewakili vertex dan setiap element dalam *linkedlist* mewakili *vertices* lain yang membentuk *edge* dengan *vertex*.

List adjacency paling sederhana membutuhkan struktur data *node* untuk menyimpan vertex dan struktur data graph untuk mengatur simpul. Kami tetap dekat dengan definisi dasar graph — kumpulan vertices dan edge {V, E}. Untuk mempermudah, kita menggunakan *graph* tidak berlabel sebagai lawan dari *graph* berlabel yaitu *vertices* diidentifikasi dengan indeksnya 0,1,2,3.

File **graph.py**

```
1.  # Adjacency List representation in Python
2.
3.  class AdjNode:
4.      def __init__(self, value):
5.          self.vertex = value
6.          self.next = None
7.
8.  class Graph:
9.      def __init__(self, num):
10.         self.V = num
11.         self.graph = [None] * self.V
12.         # Declare Array of size as Number
    of vertices, each index in array
13.         # is a vertex & each element is a
    linked list
14.
15.     # Add edges
16.     def add_edge(self, s, d):
17.         node = AdjNode(d)
18.         node.next = self.graph[s]
19.         self.graph[s] = node
20.
21.         node = AdjNode(s)
22.         node.next = self.graph[d]
23.         self.graph[d] = node
24.
25.     # Print the graph
26.     def print_agraph(self):
27.         for i in range(self.V):
28.             print("Vertex " + str(i) +
    ":", end="")
29.             temp = self.graph[i]
```

```
            while temp:
                print(" ->
    {}".format(temp.vertex), end="")
                temp = temp.next
            print(" \n")


V = 5

# Create graph and edges
graph = Graph(V)
graph.add_edge(0, 1)
graph.add_edge(0, 2)
graph.add_edge(0, 3)
graph.add_edge(1, 2)

graph.print_agraph()
"""
Vertex 0: -> 3 -> 2 -> 1

Vertex 1: -> 2 -> 0

Vertex 2: -> 1 -> 0

Vertex 3: -> 0

Vertex 4:

"""
```

BEBERAPA APLIKASI GRAF

- Lintasan terpendek (shortest path)

- Persoalan pedagang keliling (travelling salesperson problem)
- Persoalan tukang pos Cina (chinese postman problem)
- Pewarnaan graf (graph colouring)

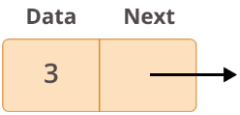
BAB VII
Linked List

Linked list adalah ordered collection dari sebuah objek. Linked list berbeda dari list dalam cara mereka menyimpan elemen dalam memori. Sementara list menggunakan blok memori yang berdekatan untuk menyimpan referensi ke datanya, linked list menyimpan referensi sebagai bagian dari elemennya sendiri.

Sebelum membahas lebih dalam tentang apa itu linked list dan bagaimana cara menggunakannya, Kita harus terlebih dahulu mempelajari bagaimana strukturnya. Setiap elemen dari linked list disebut node, dan setiap node memiliki dua field yang berbeda yaitu:

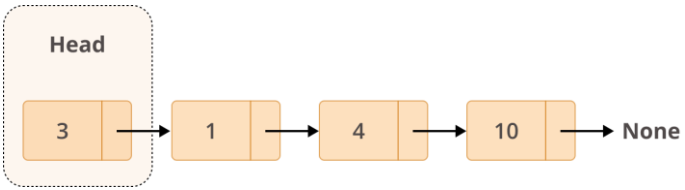
1. Data berisi nilai yang akan disimpan dalam node.
2. Next berisi referensi ke node berikutnya dalam list.

Ilustrasinya dapat dilihat sebagai berikut:



Gambar 14 Node

Linked list adalah kumpulan node. Node pertama disebut head, dan digunakan sebagai titik awal untuk setiap iterasi melalui list. Node terakhir harus memiliki referensi berikutnya yang menunjuk ke Tidak ada untuk menentukan akhir dari list.



Gambar 15 Kumpulan Node

7.1 Membuat Linked List

Linked list dibuat dengan menggunakan kelas node yang kita pelajari di bagian sebelumnya. Kita akan membuat objek Node dan membuat kelas lain untuk menggunakan objek node ini. Kita melewatkan nilai yang sesuai melalui objek node untuk mengarahkan ke elemen data berikutnya.

Program di bawah ini membuat linked list dengan tiga elemen data. Pada bagian selanjutnya kita akan melihat bagaimana melintasi linked list.

File **linkedlist.py**

```
1. # Node class
2. class Node:
3.
4.     # Function to initialize the node
   object
5.     def __init__(self, data):
6.         self.data = data  # Assign data
7.         self.next = None  # Initialize
8.         # next as null
9.
10.# Linked List class
11.
12.
13.class LinkedList:
14.
15.    # Function to initialize the Linked
16.    # List object
17.    def __init__(self):
18.        self.head = None
```

7.2 Traversing Linked List

Linked list tunggal (single linked list) dapat dilalui hanya dalam arah maju mulai dari elemen data pertama. Kita cukup mencetak nilai elemen data berikutnya dengan menugaskan pointer dari node berikutnya ke elemen data saat ini.

File **linkedlist.py**

```
1.  class Node:
2.
3.      # Function to initialise the node
    object
4.      def __init__(self, data):
5.          self.data = data  # Assign data
6.          self.next = None  # Initialize
    next as null
7.
8.
9.  # Linked List class contains a Node object
10. class LinkedList:
11.
12.     # Function to initialize head
13.     def __init__(self):
14.         self.head = None
15.
16.     # This function prints contents of
    linked list
17.     # starting from head
18.     def printList(self):
19.         temp = self.head
20.         while (temp):
21.             print(temp.data)
22.             temp = temp.next
23.
24.
25. # Code execution starts here
26. if __name__ == '__main__':
27.
28.     # Start with the empty list
29.     llist = LinkedList()
```

```
30.
31.     llist.head = Node(1)
32.     second = Node(2)
33.     third = Node(3)
34.
35.     llist.head.next = second  # Link first
    node with second
36.     second.next = third  # Link second
    node with the third node
37.
38.     llist.printList()
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
1
2
3
```

7.3 Insert pada Linked List

Memasukkan elemen dalam linked list melibatkan penugasan kembali pointer dari node yang ada ke node yang baru disisipkan. Bergantung pada apakah elemen data baru dimasukkan di awal atau di tengah atau di akhir daftar tertaut, kami memiliki skenario di bawah ini.

7.3.1 Insert pada Awal

Hal ini melibatkan penunjukan berikutnya dari simpul data baru ke kepala linked list saat ini. Jadi kepala linked list saat ini menjadi elemen data kedua dan simpul baru menjadi kepala daftar tertaut.

Pendekatan: Simpul baru selalu ditambahkan sebelum kepala linked list yang diberikan, dan node yang baru ditambahkan menjadi kepala baru dari linked list. Misalnya, jika Linked List yang diberikan adalah 10->15->20->25 dan kita menambahkan item 5 di depan, maka Linked List tersebut menjadi 5->10->15->20->25. Mari kita panggil fungsi yang menambahkan di depan daftar adalah push(). Push() harus menerima pointer ke head pointer karena push harus mengubah head pointer untuk menunjuk ke node baru.

File **linkedlist.py**

```
1.  # This function is in LinkedList class
2.  # Function to insert a new node at the
    beginning
3.  def push(self, new_data):
4.
5.      # 1 & 2: Allocate the Node &
6.      #        Put in the data
7.      new_node = Node(new_data)
8.
9.      # 3. Make next of new Node as head
```

```
10.     new_node.next = self.head
11.
12.     # 4. Move the head to point to new
    Node
13.     self.head = new_node
```

7.3.2 Insert setelah node yang diberikan

Pendekatan: Kita diberi pointer ke sebuah node, dan node baru dimasukkan setelah node yang diberikan. Ikuti langkah-langkah untuk menambahkan simpul setelah simpul yang diberikan:

1. Pertama, periksa apakah node sebelumnya adalah NULL atau tidak.
2. Kemudian, alokasikan node baru dan
3. Tetapkan data ke node baru
4. Dan kemudian jadikan simpul baru berikutnya sebagai simpul berikutnya dari simpul sebelumnya.
5. Terakhir, pindahkan simpul berikutnya dari simpul sebelumnya sebagai simpul baru.

File **linkedlist.py**

```
1.  def insertAfter(self, prev_node,
    new_data):
2.
3.      # 1. check if the given prev_node
    exists
4.      if prev_node is None:
```

```
        print("The given previous node must inLinkedList.")
        return

    # 2. Create new node &
    # 3. Put in the data
    new_node = Node(new_data)

    # 4. Make next of new Node as next of prev_node
    new_node.next = prev_node.next

    # 5. make next of prev_node as new_node
    prev_node.next = new_node
```

7.3.2 Insert di bagian akhir node

Simpul baru selalu ditambahkan setelah simpul terakhir dari linked list yang diberikan. Misalnya jika Linked List yang diberikan adalah 5->10->15->20->25 dan kita menambahkan item 30 di akhir, maka Linked List menjadi 5->10->15->20->25- >30.

Karena linked list biasanya diwakili oleh kepalanya, kita harus melintasi daftar sampai akhir dan kemudian mengubah simpul berikutnya ke simpul terakhir menjadi simpul baru.

File **linkedlist.py**

```
def append(self, new_data):
```

```
2.
3.          # 1. Create a new node
4.          # 2. Put in the data
5.          # 3. Set next as None
6.          new_node = Node(new_data)
7.
8.          # 4. If the Linked List is empty,
    then make the
9.          #    new node as head
10.         if self.head is None:
11.             self.head = new_node
12.             return
13.
14.         # 5. Else traverse till the last
    node
15.         last = self.head
16.         while (last.next):
17.             last = last.next
18.
19.         # 6. Change the next of last node
20.         last.next = new_node
```

Berikut adalah program lengkap yang menggunakan semua metode di atas untuk membuat linked list.

File **linkedlist.py**

```
1.  # Node class
2.  class Node:
3.
```

```
    # Function to initialise the node
object
    def __init__(self, data):
        self.data = data  # Assign data
        self.next = None  # Initialize
next as null


# Linked List class contains a Node object
class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None


    # Function to insert a new node at the
beginning
    def push(self, new_data):

        # 1 & 2: Allocate the Node &
        #        Put in the data
        new_node = Node(new_data)

        # 3. Make next of new Node as head
        new_node.next = self.head

        # 4. Move the head to point to new
Node
        self.head = new_node


    # This function is in LinkedList
class. Inserts a
```

```
    # new node after the given prev_node. This method is
    # defined inside LinkedList class shown above */
    def insertAfter(self, prev_node, new_data):

        # 1. check if the given prev_node exists
        if prev_node is None:
            print("The given previous node must inLinkedList.")
            return

        #  2. create new node &
        #      Put in the data
        new_node = Node(new_data)

        # 4. Make next of new Node as next of prev_node
        new_node.next = prev_node.next

        # 5. make next of prev_node as new_node
        prev_node.next = new_node


    # This function is defined in Linked List class
    # Appends a new node at the end.  This method is
    # defined inside LinkedList class shown above */
    def append(self, new_data):
```

```
57.
58.         # 1. Create a new node
59.         # 2. Put in the data
60.         # 3. Set next as None
61.         new_node = Node(new_data)
62.
63.         # 4. If the Linked List is empty,
    then make the
64.         #    new node as head
65.         if self.head is None:
66.             self.head = new_node
67.             return
68.
69.         # 5. Else traverse till the last
    node
70.         last = self.head
71.         while (last.next):
72.             last = last.next
73.
74.         # 6. Change the next of last node
75.         last.next =  new_node
76.
77.
78.     # Utility function to print the linked
    list
79.     def printList(self):
80.         temp = self.head
81.         while (temp):
82.             print(temp.data,end=" ")
83.             temp = temp.next
84.
85.
86.
87. # Code execution starts here
```

```
88. if __name__=='__main__':
89.
90.     # Start with the empty list
91.     llist = LinkedList()
92.
93.     # Insert 6.  So linked list becomes 6-
    >None
94.     llist.append(6)
95.
96.     # Insert 7 at the beginning. So linked
    list becomes 7->6->None
97.     llist.push(7);
98.
99.     # Insert 1 at the beginning. So linked
    list becomes 1->7->6->None
100.           llist.push(1);
101.
102.           # Insert 4 at the end. So
    linked list becomes 1->7->6->4->None
103.           llist.append(4)
104.
105.           # Insert 8, after 7. So linked
    list becomes 1 -> 7-> 8-> 6-> 4-> None
106.
    llist.insertAfter(llist.head.next, 8)
107.
108.           print('Created linked list is:
    ')
109.           llist.printList()
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
After insertion at head: 2 -> 1 -> NULL

After insertion at tail: 2 -> 1 -> 4 -> 5 ->
NULL

After insertion at a given position: 2 -> 1 -
> 2 -> 4 -> 5 -> 6 -> NULL
```

7.4 Search pada Linked List

Untuk mencari elemen dalam linked list, kita harus mengulangi list lengkap, membandingkan setiap node dengan data yang diinginkan, dan terus mencari sampai diperoleh kecocokan. Karena linked list tidak menyediakan akses acak, kita harus memulai pencarian dari node pertama.

Terdapat dua pendekatan yang dapat kita lakukan dalam pencarian di linked list yaitu cara iterative dan rekrusive.

7.4.1 Pendekatan Iterative

Beberapa langkah-langkah di bawah ini dapat digunakan untuk menyelesaikan masalah dengan cara pendekatan iterative:

1. Inisialisasi penunjuk node, current = head.
2. Ikuti hal ini ketika bukan NULL

- Jika nilai saat ini (mis., Current->key) sama dengan kunci yang dicari, kembalikan True.
- Jika tidak, pindah ke node berikutnya (current = current->next).

3. Jika kunci tidak ditemukan, kembalikan False

Di bawah ini adalah implementasi dari pendekatan dengan iterative.

File **Search-linkedlist.py**

```
1. class Node:
2.
3.     # Function to initialise the node
   object
4.     def __init__(self, data):
5.         self.data = data  # Assign data
6.         self.next = None  # Initialize
   next as null
7.
8. # Linked List class
9.
10.
11.class LinkedList:
12.    def __init__(self):
13.        self.head = None  # Initialize
   head as None
14.
15.    # This function insert a new node at
   the
```

```
    # beginning of the linked list
    def push(self, new_data):

        # Create a new Node
        new_node = Node(new_data)

        # 3. Make next of new Node as head
        new_node.next = self.head

        # 4. Move the head to point to new
    Node
        self.head = new_node

    # This Function checks whether the
    value
    # x present in the linked list
    def search(self, x):

        # Initialize current to head
        current = self.head

        # loop till current not equal to
    None
        while current != None:
            if current.data == x:
                return True  # data found

            current = current.next

        return False  # Data Not found


```

```
# Driver code
if __name__ == '__main__':

    # Start with the empty list
    llist = LinkedList()
    x = 21

    ''' Use push() to construct below list
        14->21->11->30->10 '''
    llist.push(10)
    llist.push(30)
    llist.push(11)
    llist.push(21)
    llist.push(14)

       # Function call
    if llist.search(x):
        print("Yes")
    else:
        print("No")
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Yes
```

7.4.2 Pendekatan Recrusive

Berikut ini merupakan langkah-langkah untuk menyelesaikan masalah dengan cara rekrusive:

1. Jika headnya adalah NULL, kembalikan false.
2. Jika head keys sama dengan X, kembalikan true;
3. Jika tidak maka mencari secara rekursif di node berikutnya.

Di bawah ini adalah implementasi dari pendekatan dengan recrusive.

File **Search-linkedlist.py**

```
1. class Node:
2.
3.     # Function to initialise
4.     # the node object
5.     def __init__(self, data):
6.         self.data = data  # Assign data
7.         self.next = None  # Initialize
   next as null
8.
9.
10.class LinkedList:
11.
12.    def __init__(self):
13.        self.head = None  # Initialize
   head as None
14.
15.    # This function insert a new node at
16.    # the beginning of the linked list
17.    def push(self, new_data):
18.
19.        # Create a new Node
```

```
        new_node = Node(new_data)

        # Make next of new Node as head
        new_node.next = self.head

        # Move the head to
        # point to new Node
        self.head = new_node

    # Checks whether the value key
    # is present in linked list

    def search(self, li, key):

        # Base case
        if(not li):
            return False

        # If key is present in
        # current node, return true
        if(li.data == key):
            return True

        # Recur for remaining list
        return self.search(li.next, key)


# Driver Code
if __name__ == '__main__':

    li = LinkedList()
```

```
51.
52.     li.push(10)
53.     li.push(30)
54.     li.push(11)
55.     li.push(21)
56.     li.push(14)
57.
58.     x = 21
59.
60.     # Function call
61.     if li.search(li.head, x):
62.         print("Yes")
63.     else:
64.         print("No")
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Yes
```

7.5 Mencari Panjang Linked List

Kita juga dapat menghitung panjang atau jumlah nodes dalam linked list. Kita bisa menggunakan dua pendekatan yaitu iterasi dan rekrusive seperti pada pencarian node di linked list.

7.5.1 Pendekatan Iterative

Beberapa langkah-langkah di bawah ini dapat digunakan untuk menyelesaikan masalah dengan cara pendekatan iterative:

1. Inisialisasi hitungan sebagai 0
2. Inisialisasi penunjuk node, current = head.
3. Ikuti hal ini jika bukan NULL
    - current = current -> next
    - Hitung kenaikan sebesar 1.
4. Kembalikan nilai jumlah node

Di bawah ini adalah implementasi dari pendekatan dengan iterative.

File **count-linkedlist.py**

```
1. class Node:
2.     # Function to initialise the node
   object
3.     def __init__(self, data):
4.         self.data = data  # Assign data
5.         self.next = None  # Initialize
   next as null
6.
7.
8. # Linked List class contains a Node object
9. class LinkedList:
10.
11.    # Function to initialize head
12.    def __init__(self):
```

```
13.         self.head = None
14.
15.     # This function is in LinkedList
    class. It inserts
16.     # a new node at the beginning of
    Linked List.
17.
18.     def push(self, new_data):
19.
20.         # 1 & 2: Allocate the Node &
21.         #     Put in the data
22.         new_node = Node(new_data)
23.
24.         # 3. Make next of new Node as head
25.         new_node.next = self.head
26.
27.         # 4. Move the head to point to new
    Node
28.         self.head = new_node
29.
30.     # This function counts number of nodes
    in Linked List
31.     # iteratively, given 'node' as
    starting node.
32.
33.     def getCount(self):
34.         temp = self.head  # Initialise
    temp
35.         count = 0  # Initialise count
36.
37.         # Loop while end of linked list is
    not reached
```

```
        while (temp):
            count += 1
            temp = temp.next
        return count


# Driver code
if __name__ == '__main__':
    llist = LinkedList()
    llist.push(1)
    llist.push(3)
    llist.push(1)
    llist.push(2)
    llist.push(1)

    # Function call
    print("Count of nodes is :",
    llist.getCount())
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Count of nodes is : 5
```

7.5.2 Pendekatan Recrusive

Beberapa langkah-langkah di bawah ini dapat digunakan untuk menyelesaikan masalah dengan cara pendekatan iterative:

1. Jika headnya NULL, kembalikan 0.

2. Jika tidak, kembalikan 1 + getCount(head->next)

Di bawah ini adalah implementasi dari pendekatan dengan iterative.

File **count-linkedlist.py**

```
1. class Node:
2.     # Function to initialise the node
   object
3.     def __init__(self, data):
4.         self.data = data  # Assign data
5.         self.next = None  # Initialize
   next as null
6.
7.
8. # Linked List class contains a Node object
9. class LinkedList:
10.
11.    # Function to initialize head
12.    def __init__(self):
13.        self.head = None
14.
15.    # This function is in LinkedList
   class. It inserts
16.    # a new node at the beginning of
   Linked List.
17.
18.    def push(self, new_data):
19.
20.        # 1 & 2: Allocate the Node &
```

```
        #     Put in the data
        new_node = Node(new_data)

        # 3. Make next of new Node as head
        new_node.next = self.head

        # 4. Move the head to point to new
    Node
        self.head = new_node

    # This function counts number of nodes
    in Linked List
    # recursively, given 'node' as
    starting node using Tail Recursion.
    def getCountRec(self, node, count=0):
        if (not node):  # Base case
            return count
        else:
            return
    self.getCountRec(node.next, count+1)

    # A wrapper over getCountRec()
    def getCount(self):
        return self.getCountRec(self.head)


# Driver code
if __name__ == '__main__':
    llist = LinkedList()
    llist.push(1)
    llist.push(3)
    llist.push(1)
```

```
    llist.push(2)
    llist.push(1)

    # Function call
    print('Count of nodes is :',
llist.getCount())
```

Hasil dari eksekusi kode tersebut dapat dilihat pada output di bawah ini.

```
Count of nodes is : 5
```

7.6 Menghapus pada Linked List

Kita juga dapat menghapus suatu node dalam linked list. Node tersebut umumnya berasal dari node yang telah kita insert pada bab sebelumnya. Terdapat 3 jenis metode penghapusan pada linked list yaitu pada bagian awal, tengah dan akhir.

7.6.1 Beginning (Awal)

Caranya yaitu arahkan head ke next node yaitu second node.

File **delete-linkedlist.py**

```
#Point head to the next node i.e. second
node
    temp = head
    head = head->next
```

```
4.
5.  #Make sure to free unused memory
6.      free(temp); or delete temp;
```

7.6.2 Middle (Tengah)

Melacak penunjuk sebelum node untuk dihapus dan penunjuk ke node untuk dihapus.

File **delete-linkedlist.py**

```
1.      temp = head;
2.      prev = head;
3.
4.      for(int i = 0; i < position; i++)
5.      {
6.          if(i == 0 && position == 1)
7.              head = head->next;
8.              free(temp)
9.          else
10.         {
11.             if (i == position - 1 && temp)
12.             {
13.                 prev->next = temp->next;
14.                 free(temp);
15.             }
16.             else
17.             {
18.                 prev = temp;
19.                 if(prev == NULL) //
    position was greater than number of nodes
    in the list
```

```
20.                     break;
21.                 temp = temp->next;
22.             }
23.         }
24.     }
```

7.6.2 End (Akhir)

Arahkan head ke elemen sebelumnya yaitu elemen kedua terakhir.

File **delete-linkedlist.py**

```
1.  #Change next pointer to null
2.      struct node *end = head;
3.      struct node *prev = NULL;
4.      while(end->next)
5.      {
6.          prev = end;
7.          end = end->next;
8.      }
9.      prev->next = NULL;
10.
11. #Make sure to free unused memory
12.     free(end); or delete end;
```

DAFTAR PUSTAKA

Baka, Benjamin. 2017. *Python Data Structures and Algorithms: Improve the Performance and Speed of Your Applications*.

Biz, Programing. 2023. "Learn Python Programming." Retrieved (https://www.programiz.com/python-programming).

Geeks, Geeks for. 2023. "Python Tutorial." Retrieved (https://www.geeksforgeeks.org/python-programming-language/).

Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser. 2013. *Data Structures & Algorithms*. Vol. 6. California: WIlley.

Jaiswal, Sejal. 2017. "Python Data Structures Tutorial." *Data Camp*.

Lambert, Kenneth A. 2014. *Fundamentals of Python Data Structures*. Boston: Cengage Learning PTR.

Tutorials Point. 2016. *Python3: Tutorialspoint*. Tutorials Point.

Purwono, S.Kom., M.Kom.
Alfian Ma Arif S.T., M.Eng.,
Dr. Ir.Iswanto, S.T., M.Eng., IPM.,

Belajar Struktur Data dengan Python

Pada buku ini dilengkapi dengan teori dan contoh implementasi struktur data menggunakan bahasa pemrograman python. Harapannya buku ini bisa menjadi panduan dasar bagi pelajar, mahasiswa atau umum yang ingin mempelajari struktur data lebih lanjut.

Penerbit UHB Press

ISBN 978-623-88102-6-0 (PDF)