Bab 4. Tipe Data Barisan

OBJEKTIF:

1. Mahasiswa mampu memahami tentang Tipe Data Barisan.
2. Mahasiswa mampu mengimplementasikan List, Tuple, dan String menggunakan bahasa pemrograman Python.
3. Mahasiswa mampu Memproses List dan Memproses String menggunakan bahasa pemrograman Pyhton.

4.1 Tipe Data Barisan

Tipe data barisan adalah tipe data yang terdiri dari kumpulan (koleksi) data. Kita menggunakan tipe data barisan untuk mengelompokkan data dalam sebuah variabel sehingga memudahkan kita untuk memanipulasi atau mengolah kelompok data tersebut. Misalkan, kita membuat program yang mengolah nilai-nilai ujian mahasiswa dalam satu kelas. Jika kita memperlakukan masing-masing nilai mahasiswa sebagai data tunggal, kita harus mempunyai variabel sebanyak jumlah mahasiswa. Misalkan terdapat lima mahasiswa, maka kita harus membuat lima variabel seperti berikut:

```
nilai_m1 = 87
nilai_m2 = 78
nilai_m3 = 63
nilai_m4 = 94
nilai_m5 = 54
```

Dengan tipe data barisan kita dapat menyimpan nilai-nilai semua mahasiswa dalam satu variabel, seperti terlihat pada statement berikut:

```
nilai_kelas = [87, 78, 63, 94, 54]
```

Terdapat sejumlah tipe data barisan dalam Python, kita akan membahas dua diantaranya:

- `list`
- `tuple`

Selain kedua tipe data barisan, string juga dapat dipandang sebagai tipe data yang menyimpan barisan karakter-karakter sehingga mempunyai kemiripan dengan tipe data barisan.

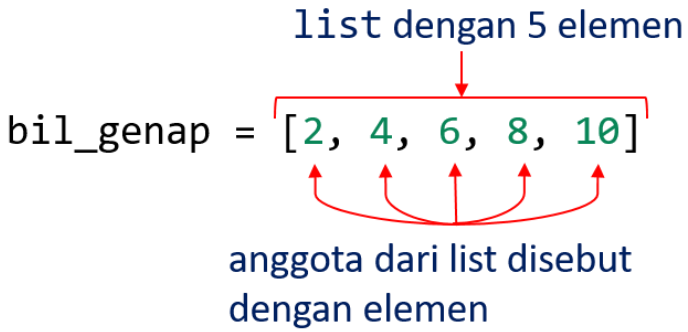
Pada bagian ini kita akan membahas: `list`, `tuple`, dan `string`.

4.2 List

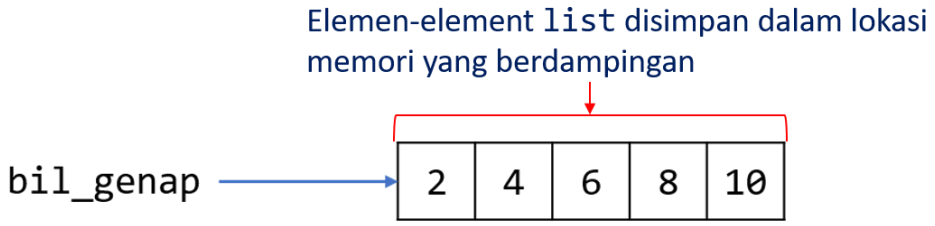
List adalah barisan data dari tipe data apapun. Kita membuat list dengan menuliskan nilai-nilai yang dipisahkan tanda koma di dalam kurung kotak seperti contoh berikut:

```
bil_genap = [2, 4, 6, 8, 10]
```

Anggota-anggota dari list disebut dengan elemen. Pada contoh di atas 2, 4, 6, 8, dan 10 adalah elemen-elemen dari list `bil_genap`.



Elemen-elemen dari list disimpan dalam lokasi memori yang berdampingan. Model memori dari list diilustrasikan oleh gambar berikut:



Selain mempunyai elemen-elemen berupa tipe numerik, kita juga bisa mempunyai sebuah list dengan elemen-elemennya berupa string, seperti contoh berikut:

```
nama = ['Budi', 'Johan', 'Wati', 'Anisa', 'Joko']
```

Elemen-elemen dari list dapat juga berbeda tipe:

```
info = ['Anisa', 20, 3.85]
```

Meskipun kita dapat mempunyai list dengan elemen-elemen berbeda tipe, umumnya list digunakan untuk elemen-elemen dengan tipe sama.

Fungsi `print()` dapat dipanggil dengan argumen berupa sebuah list untuk menampilkan elemen-elemen dari list tersebut:

```
>>> bilangan = [5, 10, 15, 20]
>>> print(bilangan)
[5, 10, 15, 20]
```

Kita bisa menggunakan fungsi `type()` untuk melihat tipe data list, seperti contoh berikut:

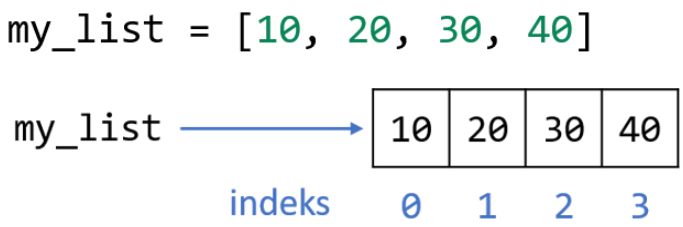
```
>>> bil_ganjil = [1, 3, 5, 7]
>>> type(bil_ganjil)
<class 'list'>
```

List dapat terdiri dari nol elemen (disebut dengan list kosong) ataupun satu elemen saja, seperti contoh berikut:

```
>>> list_kosong = []
>>> print(list_kosong)
[]
>>> list_satu_el = ['satu']
>>> print(list_satu_el)
['satu']
```

4.2.1 Indeks

Setiap elemen dalam sebuah list mempunyai nomor urut yang menandakan posisinya dalam list tersebut. Nomor urut ini disebut dengan **indeks**. Indeks dimulai dari 0 yang menandakan elemen pertama lalu dilanjutkan dengan indeks 1 yang menandakan elemen kedua dan seterusnya. Gambar berikut mengilustrasikan contoh list beserta indeks dari elemen-elemennya:



Kita dapat mengakses elemen individu dalam list dengan menuliskan indeks dari elemen dalam tanda kurung kotak setelah nama variabel yang menyimpan list, seperti contoh berikut:

```
>>> my_list = [10, 20, 30, 40]
>>> my_list[0]
10
>>> my_list[1]
20
>>> my_list[3]
40
```

Pada contoh sesi interaktif di atas, ekspresi `my_list[0]` mengakses elemen indeks 0 dari `my_list` yang bernilai `10`, ekspresi `my_list[1]` mengakses elemen indeks 1 dari `my_list` yang bernilai `20`, dan seterusnya.

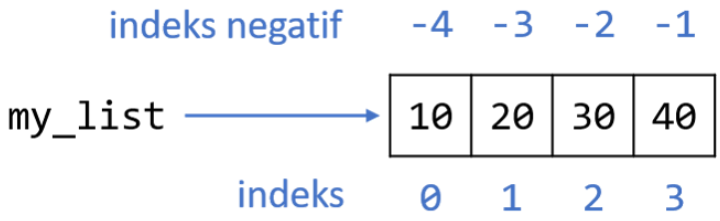
Secara umum syntax untuk mengakses elemen list adalah seperti berikut:

```
<list>[<indeks>]
```

Perlu diperhatikan bahwa indeks dari elemen terakhir dari sebuah list adalah banyaknya elemen dikurangi 1 (karena indeks dimulai dari nol). Jika kita mencoba menggunakan indeks yang melebihi indeks elemen terakhir, kita akan mendapatkan *IndexError*, seperti terlihat pada contoh sesi interaktif berikut:

```
>>> my_list = [10, 20, 30, 40]
>>> my_list[4]
Traceback (most recent call last):
 File "<pyshell#81>", line 1, in <module>
  my_list[4]
IndexError: list index out of range
```

Kita juga dapat mengakses elemen-elemen individu menggunakan indeks negatif. Indeks negatif -1 menandakan elemen terakhir, indeks -2 menandakan indeks elemen kedua terakhir, dst. Gambar berikut mengilustrasikan contoh sebuah list dan indeks negatifnya:



Sesi berikut mencontohkan penggunaan indeks negatif untuk mengakses elemen-elemen dari sebuah list:

```
>>> my_list = [10, 20, 30, 40]
>>> my_list[-1]
40
>>> my_list[-2]
30
>>> my_list[-3]
20
>>> my_list[-4]
10
```

4.2.2 Mengiterasi List

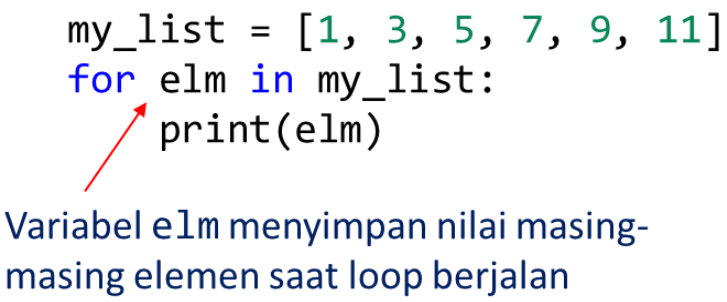
Semua data berbentuk barisan dapat di-iterasi menggunakan loop `for`. Kita dapat mengiterasi list, seperti contoh berikut:

```
>>> my_list = [1, 3, 5, 7, 9, 11]
>>> for elm in my_list:
        print(elm)
```

Kode di atas menghasilkan output berikut:

```
1
3
5
7
9
11
```

Gambar berikut menjelaskan loop `for` di atas:



Berikut adalah contoh lain dari penggunaan loop `for` untuk mengiterasi list:

```
names = ['Budi', 'Johan', 'Wati', 'Anisa', 'Joko']
for name in names:
  print(name)
```

Output dari kode di atas:

```
Budi
Johan
Wati
Anisa
Joko
```

4.2.3 Fungsi `len()`

Kita dapat menggunakan fungsi built-in `len()` untuk mendapatkan **panjang** list (banyaknya elemen), seperti terlihat pada contoh sesi interaktif berikut:

```
>>> my_list = [10, 20, 30, 40]
>>> panjang = len(my_list)
>>> print(panjang)
4
```

```
>>> nama = ['Budi', 'Johan', 'Wati', 'Anisa', 'Joko']
>>> len(nama)
5
```

Fungsi `len()` umumnya digunakan dalam loop untuk mencegah *IndexError* saat mengiterasi list:

```
my_list = [10, 20, 30, 40]
index = 0
while index < len(my_list):
    print(my_list[index])
    index += 1
```

Output dari kode di atas:

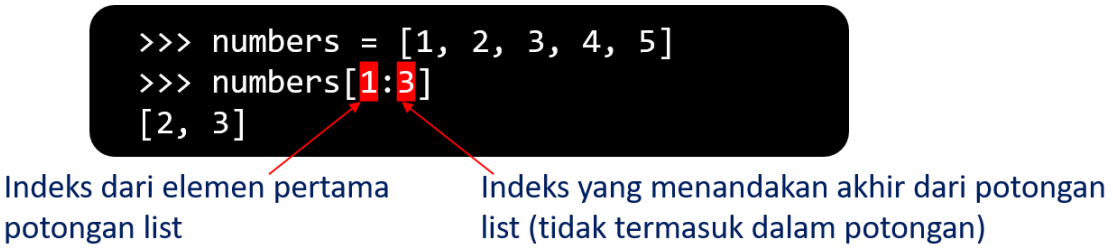
```
10
20
30
40
```

4.2.4 Memotong List

Kita dapat mengambil beberapa elemen list yang berdampingan dengan melakukan operasi pemotongan list, seperti terlihat pada contoh berikut:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[1:3]
```

Ekspresi `numbers[1:3]` memberikan sebuah list baru yang elemen-elemennya adalah elemen indeks 1, 2, dan 3 dari list `numbers`. Gambar berikut menjelaskan sesi interaktif di atas:



Bentuk umum syntax penulisan ekspresi operasi pemotongan list adalah seperti berikut:

```
<list>[<awal>:<akhir>]
```

Dimana `<awal>` adalah indeks awal dari potongan list yang ingin didapatkan dan `<akhir>` adalah indeks batas akhir dari potongan list (elemen indeks `<akhir>` tidak termasuk dalam potongan). Ekspresi operasi pemotongan list menghasilkan sebuah list baru dengan elemen-elemen dimulai dari elemen pada indeks `<awal>` dari `<list>` sampai dengan indeks `<akhir> - 1` dari `<list>`.

Contoh lain pemotongan list:

```
>>> hari = ['Senin', 'Selasa', 'Rabu', 'Kamis', 'Jumat', 'Sabtu', 'Minggu']
>>> hari_kerja = hari[0:5]
>>> print(hari_kerja)
['Senin', 'Selasa', 'Rabu', 'Kamis', 'Jumat']
```

Jika kita tidak menuliskan `<awal>` pada ekspresi pemotongan list, indeks 0 akan digunakan sebagai `<awal>`, seperti contoh berikut:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[:3]
[1, 2, 3]
```

Jika kita tidak menuliskan `<akhir>` pada ekspresi pemotongan list, maka panjang dari list akan digunakan sebagai `<akhir>`, seperti contoh berikut:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[2:]
[3, 4, 5]
```

Jika kita tidak menuliskan `<awal>` dan `<akhir>` pada ekspresi pemotongan list, maka kita akan mendapatkan salinan keseluruhan list, seperti contoh berikut:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[:]
[1, 2, 3, 4, 5]
```

Kita dapat menambahkan step (langkah) pada ekspresi pemotongan list untuk mendapatkan potongan list yang melewati beberapa elemen, seperti contoh berikut:

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

Step 2 berarti melangkah setiap 2 elemen (melewati 1 elemen per langkah).

Kita juga dapat menggunakan indeks negatif pada ekspresi pemotongan list:

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

4.2.5 Mengkonkatenasi List

Mengkonkatenasi berarti menyambung. Kita dapat mengkonkatenasi dua buah list menggunakan operator `+`, seperti contoh berikut:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [5, 6, 7, 8]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Kita dapat mengekstensi sebuah list dengan list lain dengan konkatenasi:

```
>>> my_list1 = [1, 2, 3, 4]
>>> my_list1 = my_list1 + [5, 6, 7, 8]
>>> print(my_list1)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Kita dapat menggunakan operator augmented assignment untuk mengekstensi list seperti yang kita lakukan pada contoh sebelumnya:

```
>>> my_list1 = [1, 2, 3, 4]
>>> my_list1 += [5, 6, 7, 8]
>>> print(my_list1)
[1, 2, 3, 4, 5, 6, 7, 8]
```

4.2.6 Melipatgandakan List

Operator `*` jika digunakan dengan kombinasi operand berupa list dan operand berupa integer, dapat digunakan untuk melipatgandakan sebuah list. Pada contoh di berikut, kita melipatgandakan list sebanyak tiga kali:

```
>>> list4 = list1 * 3
>>> print(list4)
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

Melipatgandakan list umumnya digunakan untuk menginisialisasi sebuah list dengan panjang dan nilai awal tertentu, seperti pada contoh berikut:

```
>>> kelompok_nilai = [0] * 15
>>> print(kelompok_nilai)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Dengan menggunakan operator *, kita tidak perlu menuliskan 0 sebanyak 15 kali.

4.2.7 List Bersifat *Mutable*

Mutable berarti dapat berubah. List bersifat mutable berarti elemen-elemen dari list dapat diubah nilainya. Kita mengubah nilai elemen dari list dengan memilih elemen yang ingin diubah menggunakan indeks dan menugaskannya dengan nilai baru. contoh berikut mengubah nilai elemen dengan indeks 0 dari `my_list` menjadi 22:

```
>>> my_list = [32, 88, 43]
>>> my_list[0] = 22
>>> print(my_list)
[22, 88, 43]
```

Pada pembahasan loop `for`, kita menggunakan fungsi `range()` untuk menggenerasi barisan integer. Barisan integer yang digenerasi oleh fungsi `range()` bukanlah berbentuk list, seperti terlihat pada contoh sesi interaktif berikut:

```
>>> bar_int = range(10)
>>> type(bar_int)
<class 'range'>
>>> print(bar_int)
range(0, 10)
```

Kita dapat menggunakan fungsi built-in `list()` untuk mengubah barisan integer yang digenerasi oleh fungsi `range()` menjadi sebuah list:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
```

4.2.8 Method-method pada List

Semua tipe data dalam Python adalah object. Setiap object mempunyai attribute dan method. Attribute adalah data yang terasosiasikan dengan object. Method adalah fungsi yang terasosiasikan dengan object. Method dari object umumnya melakukan operasi terhadap attribute-attribute dari object. Seperti fungsi, method dapat juga menerima argumen dan mengembalikan nilai.

List adalah object, dimana attribute dari object list adalah elemen-elemen dari list tersebut. Method-method dari object list akan mengubah elemen-elemen dari list tersebut.

Kita memanggil method dari list menggunakan notasi dot dengan syntax berikut:

```
<list>.<nama_method>(<argumen1>, <argumen2>, ...)
```

Misalkan, list mempunyai method bernama `sort` yang mengurutkan elemen-elemen pada list dari nilai terendah ke nilai tertinggi:

```
>>> my_list = [88, 43, 96, 34]
>>> my_list.sort()
>>> print(my_list)
[34, 43, 88, 96]
```

Setelah menjalankan method sort pada `my_list`, elemen-elemen dari `my_list` menjadi berurut dari nilai terkecil ke nilai terbesar.

Tabel berikut mendaftar beberapa method-method dari list:

Method	Keterangan
`<list>.append(x)`	Menambahkan elemen `x` ke akhir dari list
`<list>.sort()`	Mengurutkan elemen-elemen list
`<list>.reverse()`	Membalik urutan elemen-elemen list
`<list>.index(x)`	Mengembalikan indeks dari kemunculan pertama dari `x`
`<list>.insert(i,x)`	Memasukkan `x` ke list pada indeks `i`
`<list>.count(x)`	Mengembalikan banyaknya kemunculan `x` dalam list
`<List>.remove(x)`	Menghapus kemunculan pertama dari `x` dalam list
`<list>.pop(i)`	Menghapus elemen dengan indeks `i` dari list dan mengembalikan nilai elemen tersebut

`append()`

Append berarti tambahkan di akhir. Method `append()` menambahkan sebuah elemen ke akhir dari list. Contoh penggunaan method `append()`:

```
>>> my_list = [1, 3, 5, 7, 9]
>>> my_list.append(11)
>>> print(my_list)
[1, 3, 5, 7, 9, 11]
```

`reverse()`

Reverse berarti kebalikan. Method `reverse()` membalik urutan elemen-elemen list. Contoh penggunaan method `reverse()`:

```
>>> my_list = [1, 3, 5, 7, 9]
>>> my_list.reverse()
>>> print(my_list)
[9, 7, 5, 3, 1]
```

`index()`

Index berarti indeks. Method `index()` mengembalikan nomor indeks dari kemunculan pertama dari argument yang diberikan dalam list. Contoh penggunaan method `index()`:

```
>>> list1 = ['satu', 'dua', 'tiga']
>>> list1.index('satu')
0
```

`insert()`

Insert berarti masukkan. Method `insert()` menerima dua argument: sebuah nomor indeks dan sebuah nilai. Method `insert()` menyisipkan nilai argument kedua yang diberikan pada indeks yang diberikan sebagai argument pertama. Contoh penggunaan method `insert()`:

```
>>> nama = ['Jimi', 'Hendrik', 'Billy']
>>> nama.insert(1, 'Jordi')
>>> print(nama)
['Jimi', 'Jordi', 'Hendrik', 'Billy']
```

`count()`

Count berarti hitung. Method `count()` menghitung banyaknya elemen dengan nilai yang sama dengan nilai argument. Contoh penggunaan method `count()`:

```
>>> nomor_acak = [2, 4, 3, 9, 11, 2, 6, 4, 5, 9]
>>> nomor_acak.count(2)
2
>>> nomor_acak.count(4)
2
```

`remove()`

Remove berarti buang. Method `remove()` digunakan untuk menghapus elemen pertama dalam list yang nilainya sama dengan nilai argument. Contoh penggunaan method `remove()`:

```
>>> nomor_acak = [2, 4, 3, 9, 11, 2, 6, 4, 5, 9]
>>> nomor_acak.remove(2)
>>> print(nomor_acak)
[4, 3, 9, 11, 2, 6, 4, 5, 9]
>>> nomor_acak.remove(9)
>>> print(nomor_acak)
[4, 3, 11, 2, 6, 4, 5, 9]
```

Perhatikan pada statement ke-3 dari sesi interaktif di atas:

```
nomor_acak.remove(9)
```

Statement ini hanya menghapus elemen dengan nilai 9 yang pertama (elemen terakhir yang juga bernilai 9 tidak dihapus).

Untuk menghapus sebuah elemen list dengan indeks tertentu, kita dapat menggunakan statement `del`, seperti contoh berikut yang akan menghapus elemen dengan indeks 2 dari list yang direferensikan oleh variabel `my_list`:

```
>>> my_list = [1, 2, 3, 4, 5]
>>> del my_list[2]
>>> print(my_list)
[1, 2, 4, 5]
```

Kita juga dapat menggunakan statement `del` dengan ekspresi pemotongan list untuk menghapus potongan list, seperti contoh berikut yang akan menghapus potongan list dari indeks 1 s.d indeks 2 dari list yang direferensikan oleh variabel `my_list`:

```
>>> my_list = [1, 2, 3, 4, 5]
>>> del my_list[1:3]
>>> print(my_list)
[1, 4, 5]
```

`pop()`

Pop berarti ambil. Method `pop()` menghapus elemen dengan indeks sesuai argument. Contoh penggunaan method `pop()`:

```
>>> nama = ['Jimi', 'Hendrik', 'Billy']
>>> nama.pop(2)
'Billy'
>>> print(nama)
['Jimi', 'Hendrik']
```

4.2.9 Menyalin List

Saat kita menyalin tipe data tunggal, nilai yang disimpan oleh variabel pada ruas kanan akan disalin dan ditugaskan ke variable pada ruas kiri. Perhatikan sesi interaktif berikut:

```
>>> y = 3
>>> x = y
>>> y = 5
>>> print(y)
5
>>> print(x)
3
```

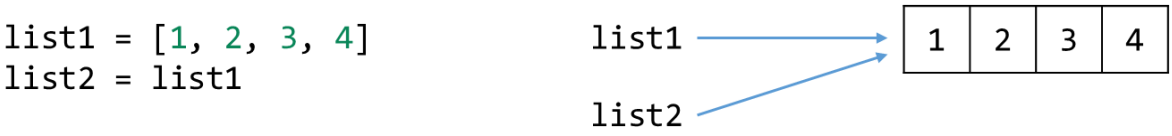
Statement `x = y` menyalin nilai yang disimpan variabel `y` yaitu 3 dan menugaskannya ke variabel `x`. Jika kita mengubah nilai variabel `y`, dalam contoh di atas kita mengubah ke nilai 5 (`y = 5`), maka nilai `x` tidak akan ikut berubah.

Menyalin list berbeda dengan menyalin data tunggal. Perhatikan sesi interaktif berikut:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = list1
>>> list1[0] = 99
>>> print(list1)
[99, 2, 3, 4]
>>> print(list2)
[99, 2, 3, 4]
```

Statement `list2 = list1` **tidak** menyalin isi `list1` dan menugaskannya ke `list1`, tetapi statement tersebut menyalin alamat memori tempat disimpannya isi dari `list1`. Sehingga setelah statement ini, `list2` akan merujuk list yang sama seperti yang dirujuk oleh `list1`. Untuk melihat `list1` dan `list2` merujuk ke sebuah list yang sama, misalkan kita mengubah nilai elemen dari salah satu list, pada contoh di atas kita statement `list1[0] = 99` mengubah nilai elemen indeks 0 dari `list1` ke 99. Jika kita lihat isi `list1` dan `list2` setelah statement ini, maka kita akan melihat `list1` dan `list2` mempunyai isi list yang sama.

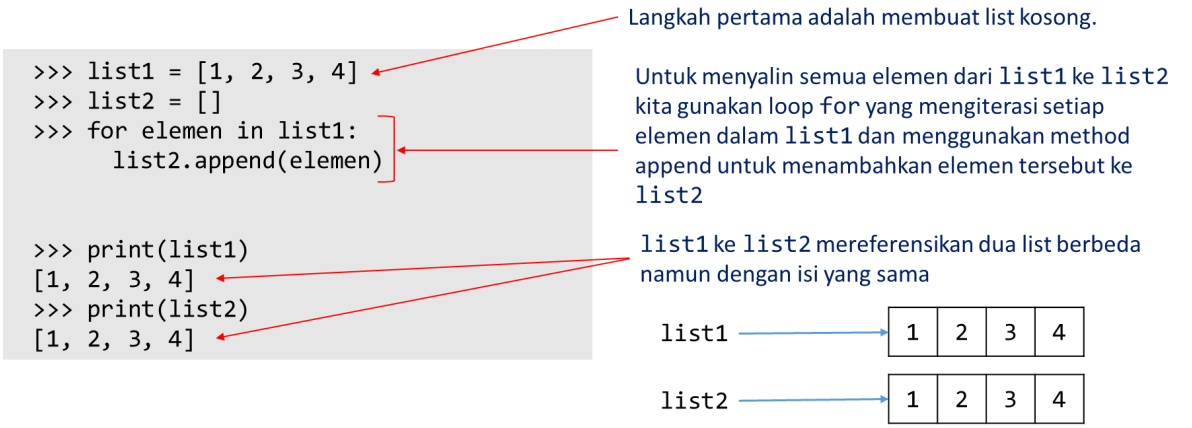
Gambar berikut mengilustrasikan model memori yang dihasilkan oleh statement `list2 = list1`:



Bagaimana kita menyalin isi list sehingga kita mempunyai dua list yang berbeda tetapi dengan isi yang sama? Satu cara untuk menyalin elemen-elemen dari sebuah list ke list lain, sehingga `list1` dan `list2` mereferensikan dua list berbeda namun dengan elemen-elemen yang sama adalah dengan menggunakan loop `for` seperti dicontohkan pada sesi interaktif berikut:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = []
>>> for elemen in list1:
    list2.append(elemen)
>>> print(list1)
[1, 2, 3, 4]
>>> print(list2)
[1, 2, 3, 4]
```

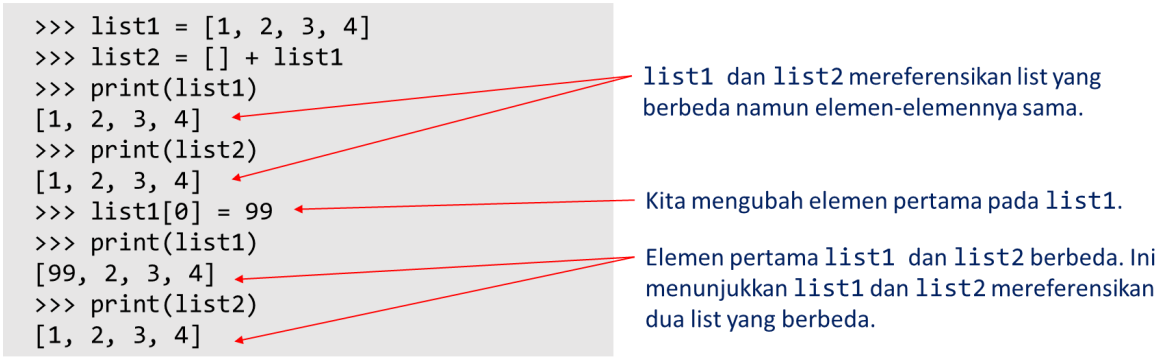
Sesi interaktif di atas menghasilkan dua buah list yang berbeda namun mempunyai isi yang sama. Gambar berikut menjelaskan sesi interaktif di atas:



Cara lain menyalin elemen-elemen sebuah list ke list baru yang lebih mudah:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [] + list1
>>> print(list1)
[1, 2, 3, 4]
>>> print(list2)
[1, 2, 3, 4]
>>> list1[0] = 99
>>> print(list1)
[99, 2, 3, 4]
>>> print(list2)
[1, 2, 3, 4]
```

Gambar berikut menjelaskan sesi interaktif di atas:

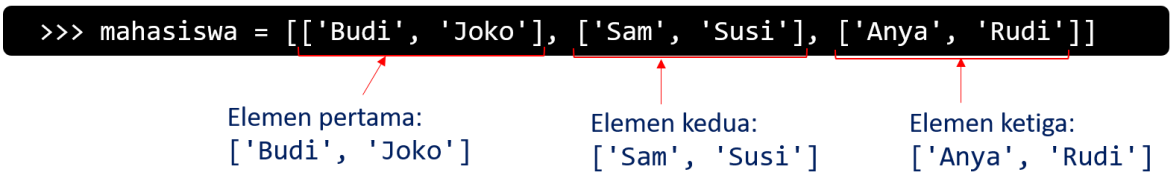


4.2.10 List Dua Dimensi

Elemen dalam sebuah list dapat berupa tipe data apapun, termasuk sebuah list juga:

```
>>> mahasiswa = [['Budi', 'Joko'], ['Sam', 'Susi'], ['Anya', 'Rudi']]
```

Gambar berikut menjelaskan elemen-elemen dari list di atas:



List yang elemennya berupa list juga disebut dengan list dua dimensi. Kita dapat membayangkan list dua dimensi sebagai tabel. List `mahasiswa` pada contoh di atas dapat dibayangkan sebagai tabel seperti berikut:

	Kolom 0	Kolom 1
Baris 0	`'Budi'`	`'Joko'`
Baris 1	`'Sam'`	`'Susi'`
Baris 2	`'Anya'`	`'Rudi'`

Kita mengakses baris dari elemen list dua dimensi seperti berikut:

```
>>> mahasiswa[0]
['Budi', 'Joko']
>>> mahasiswa[1]
['Sam', 'Susi']
>>> mahasiswa[2]
['Anya', 'Rudi']
```

Kita mengakses elemen-elemen pada setiap baris:

Baris 0:

```
>>> mahasiswa[0][0]
'Budi'
>>> mahasiswa[0][1]
'Joko'
```

Baris 1:

```
>>> mahasiswa[1][0]
'Sam'
>>> mahasiswa[1][1]
'Susi'
```

Baris 2:

```
>>> mahasiswa[2][0]
'Anya'
>>> mahasiswa[2][1]
'Rudi'
```

4.3 Tuple

Tuple adalah tipe data barisan yang mirip dengan list. Untuk membuat tuple, kita menuliskan nilai-nilai yang dipisahkan koma di dalam tanda kurung:

```
my_tuple = (10, 20, 30, 40)
```

Perbedaan tuple dengan list adalah tuple bersifat *immutable*. Immutable berarti elemen-elemen dari tuple tidak bisa berubah. Semua operasi-operasi yang bisa dilakukan terhadap list dapat dilakukan terhadap tuple, kecuali yang mengubah elemen-elemen.

Seperti pada list, kita dapat melakukan pengindeksan dan pemotongan terhadap tuple:

```
>>> my_tuple1 = ('satu', 'dua', 'tiga', 'empat', 'lima')
>>> my_tuple1[4]
'lima'
>>> my_tuple1[-2]
'empat'
>>> my_tuple1[1:4]
('dua', 'tiga', 'empat')
>>> my_tuple1[1::2]
('dua', 'empat')
```

Namun kita tidak dapat melakukan operasi yang memodifikasi elemen-elemen tuple:

```
>>> my_tuple2 = (1, 'dua', 3, 'empat')
>>> my_tuple2[1] = 2
Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    my_tuple2[1] = 2
TypeError: 'tuple' object does not support item assignment
```

Perhatikan statement kedua dari sesi interaktif di atas, `my_tuple[1] = 2`, kita mencoba untuk mengubah nilai elemen indeks 1 dari tuple `my_tuple2`. Statement ini akan menghasilkan error karena sifat tuple yang immutable.

Tuple mendukung operasi-operasi berikut:

- Pengindeksan (untuk mendapatkan nilai elemen individu saja tapi tidak bisa untuk penugasan ulang)
- Ekspresi Pemotongan
- Operator + dan *
- Fungsi `len()`
- Method-method pada list yang tidak mengubah nilai-nilai elemen atau menambahkan elemen: method count dan method indeks

Tuple tidak mendukung method-method list yang mengubah nilai-nilai elemen seperti: `append()`, `remove()`, `insert()`, `reverse()`, `sort()`, dan `pop()`.

Kita menggunakan tuple dibandingkan list ketika:

- Memroses banyak data dan data tersebut tidak perlu dimodifikasi. Karena tuple bersifat immutable, maka pemrosesan tuple lebih cepat dibandingkan dengan pemrosesan list.
- Jika kita ingin data kita tidak dimodifikasi sepanjang program berjalan. Menggunakan tuple akan memastikan data-data tidak berubah.

Kita dapat menggunakan fungsi built-in `list()` untuk mengkonversi sebuah tuple menjadi sebuah list dan fungsi built-in `tuple()` untuk mengkonversi sebuah list menjadi tuple:

```
>>> number_tuple = (1, 2, 3)
>>> number_list = list(number_tuple)
>>> print(number_list)
[1, 2, 3]
```

```
>>> str_list = ['satu', 'dua', 'tiga']
>>> str_tuple = tuple(str_list)
>>> print(str_tuple)
('satu', 'dua', 'tiga')
```

4.4 Memroses List

Sejauh ini kita telah melihat teknik-teknik untuk bekerja dengan list. Pada bagian ini kita akan melihat beberapa cara program dapat memproses data yang disimpan dalam list:

- Menjumlahkan nilai-nilai dalam list
- Menghitung rata-rata dari nilai-nilai dalam list
- Memberikan list sebagai argument ke sebuah fungsi
- Mengembalikan referensi ke list dari sebuah fungsi
- Memroses list dua dimensi

4.4.1 Menjumlahkan Nilai-nilai Elemen List

Untuk mencari total penjumlahan dari nilai-nilai dalam list, kita menggunakan sebuah `loop` dan sebuah variabel akumulator. Program berikut menjumlahkan nilai-nilai elemen dalam list:

```
# total_list.py
# Program ini menghitung total nilai-nilai dalam sebuah list
def main():
    # Buat list
    numbers = [2, 4, 6, 8, 10]

    # Buat sebuah variabel sebagai akumulator
    total = 0

    # Hitung total dari elemen list
    for value in numbers:
        total += value

    # Tampilkan total dari elemen-elemen list
    print('Total dari elemen-elemen adalah', total)

# Panggil fungsi main
main()
```

Output:

```
Total dari elemen-elemen adalah 30
```

4.4.2 Mencari Rata-rata dari Nilai-nilai Elemen List

Untuk mencari rata-rata dari nilai-nilai dalam list maka hal yang harus kita lakukan yaitu:

- Hitung total elemen-elemen list
- Bagi total dengan banyak elemen dalam list

Program berikut menghitung rata-rata elemen-elemen dalam sebuah list:

```
# average_list.py
# Program ini menghitung rata-rata nilai-nilai dalam  list
def main():
    # Buat list
    skor = [2.5, 8.3, 6.5, 4.0, 5.2]
```

```
    # Buat sebuah variabel sebagai akumulator
    total = 0.0

    # Hitung total dari elemen list
    for value in skor:
        total += value

    # Hitung rata-rata
    average = total / len(skor)

    # Tampilkan total dari elemen-elemen list
    print('Rata-rata dari elemen-elemen adalah', average)

# Panggil fungsi main
main()
```

Output:

```
Rata-rata dari elemen-elemen adalah 5.3
```

4.4.3 Memberikan List Sebagai Argumen ke Fungsi

Kita dapat membuat sebuah fungsi yang menerima sebuah list sebagai argumen.

```
# fungsi_total.py
# Program ini menggunakan sebuah fungsi untuk menghitung
# total dari nilai-nilai dalam list
def get_total(nilai_list):
    # Buat sebuah variabel sebagai akumulator
    total = 0

    # Hitung total dari elemen-elemen list
    for num in nilai_list:
        total += num

    # Kembalikan nilai total
    return total

def main():
    # Buat sebuah list
    numbers = [2, 4, 6, 8, 10]

    # Tampilkan total dari elemen
    print('Total dari elemen list: ', get_total(numbers))

# Panggil fungsi main
main()
```

Output:

```
Total dari elemen list:  30
```

Pada contoh program `fungsi_total.py` di atas, kita mendefinisikan sebuah fungsi bernama `get_total` yang menerima sebuah list sebagai argumen.

4.4.4 Mengembalikan Referensi ke List dari Sebuah Fungsi

Fungsi juga dapat mengembalikan sebuah referensi ke list:

```
# return_list.py
# Program ini menggunakan sebuah fungsi yang membuat list
# dan mengembalikan referensi ke list

# Fungsi get_values menerima rangkaian angka
# dari pengguna dan menyimpannya dalam sebuah list.
# Fungsi ini kemudian mengembalikan referensi
# ke list tersebut.
def get_values():
    # Buat list kosong
    values = []

    # Buat sebuah variabel untuk mengendalikan loop
    lanjut = 'y'

    # Ambil nilai-nilai dari pengguna dan tambahkan ke list
    while lanjut == 'y':
        # Ambil sebuah angka dan tambahkan ke list
        num = int(input('Masukkan sebuah angka: '))
        values.append(num)

        # Lanjut input
        print('Apakah Anda ingin menambahkan angka?')
        lanjut = input('y = ya, lainnya = tidak: ')
        print()

    # kembalikan list yang menyimpan input
    return values

def main():
    # Ambil sebuah list yang menyimpan nilai-nilai
    numbers = get_values()

    # Tampilkan nilai-nilai dalam list
    print('Angka-angka dalam list: ')
    print(numbers)

# Panggil fungsi main
main()
```

Output:

```
Masukkan sebuah angka: 3
Apakah Anda ingin menambahkan angka?
y = ya, lainnya = tidak: y

Masukkan sebuah angka: 56
Apakah Anda ingin menambahkan angka?
y = ya, lainnya = tidak: y

Masukkan sebuah angka: 4
Apakah Anda ingin menambahkan angka?
```

```
y = ya, lainnya = tidak:

Angka-angka dalam list:
[3, 56, 4]
```

4.4.5 Memroses List Dua Dimensi

Misalkan kita membuat program yang menghitung rata-rata nilai tiga ujian dari empat mahasiswa dalam sebuah kelas. Kita dapat menyimpan nilai-nilai tersebut ke dalam list dua dimensi lalu memrosesnya untuk mendapatkan nilai rata-rata. Program berikut mencontohkan pemrosesan list dua dimensi:

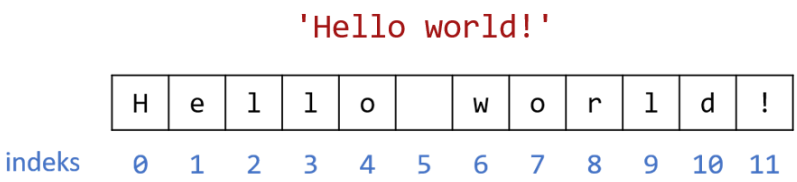
```
# average_2d_list.py
# Program ini menggunakan list 2d untuk menyimpan
# 3 nilai ujian dari setiap mahasiswa dalam satu kelas
def main():

    # Constant untuk Jumlah Ujian (Kolom)
    # dan Jumlah Mahasiswa (Baris)
    JUMLAH_UJIAN = 3
    JUMLAH_MHS = 4

    nilai_ujian = [[84.5, 77.4, 69.3],
                   [74.3, 54.2, 98.4],
                   [87.4, 98.3, 96.3],
                   [43.5, 95.4, 100]]

    # Iterasi setiap mahasiswa (baris)
    for mhs in range(JUMLAH_MHS):

        # Akumulator nilai ujian
        total_nilai = 0

        # Iterasi nilai ujian dari mahasiswa mhs (kolom)
        for ujian in range(JUMLAH_UJIAN):
            total_nilai += nilai_ujian[mhs][ujian]

        # Hitung rata-rata ujian
        rerata = total_nilai / JUMLAH_UJIAN

        # Tampilkan rata-rata nilai ujian setiap mahasiswa
        print(f'Rata-rata ujian {mhs + 1}: {rerata:.2f}')

# Panggil fungsi main
main()
```

Output:

```
Rata-rata ujian 1: 77.07
Rata-rata ujian 2: 75.63
Rata-rata ujian 3: 94.00
Rata-rata ujian 4: 79.63
```

4.5 String

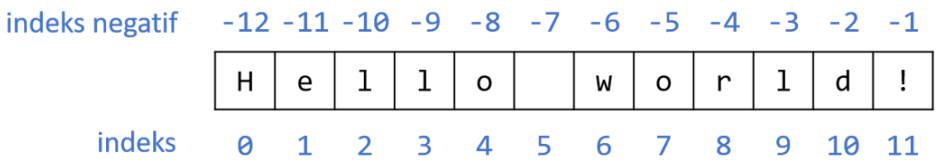
String dapat dipandang sebagai barisan dari karakter-karakter. Seperti list, karakter-karakter dari string memiliki indeks. Gambar berikut mengilustrasikan sebuah string dan indeks dari karakter-karakternya:



Kita dapat mengakses karakter-karakter individu dalam string dengan pengindeksan:

```
>>> my_string = 'Hello world!'
>>> my_string[0]
'H'
>>> my_string[3]
'l'
>>> my_string[5]
' '
>>> my_string[11]
'!'
```

Karakter-karakter pada string juga dapat diakses menggunakan indeks negatif:



Indeks negatif memudahkan kita untuk mengakses karakter terakhir tanpa perlu mencari panjang string terlebih dahulu:

```
>>> my_string = 'Hello world!'
>>> my_string[-1]
'!'
>>> my_string[-5]
'o'
```

Eksepsi *IndexError* akan muncul jika kita mencoba menggunakan indeks yang melewati karakter terakhir dari string:

```
>>> my_string = 'Hello world!'
>>> my_string[12]
Traceback (most recent call last):
  File "<pyshell#304>", line 1, in <module>
    my_string[12]
IndexError: string index out of range
```

Begitu juga dengan indeks negatif, jika kita menggunakan indeks negatif yang melewati batas karakter dari string, kita akan mendapatkan *IndexError*:

```
>>> my_string = 'Hello world!'
>>> my_string[-13]
Traceback (most recent call last):
  File "<pyshell#98>", line 1, in <module>
    my_string[-13]
IndexError: string index out of range
```

Karena string berbentuk barisan data, kita dapat melakukan iterasi terhadap string menggunakan loop `for`:

```
my_string = 'Hello, world!'
for chr in my_string:
    print(chr)
```

Output dari kode di atas:

```
H
e
l
l
o
,

w
o
r
l
d
!
```

4.5.1 Fungsi `len()`

Fungsi `len()` dapat digunakan untuk mendapatkan panjang (banyak karakter) dari sebuah string:

```
>>> kota = 'Bandung'
>>> panjang_str = len(kota)
>>> print(panjang_str)
7
```

Fungsi `len()` berguna untuk mencegah loop mengiterasi melewati akhir dari string:

```
kota = 'Bandung'
index = 0
while index < len(kota):
    print(kota[index])
    index += 1
```

Output dari kode di atas:

```
B
a
n
d
u
n
g
```

4.5.2 Memotong String

Seperti pada list, kita dapat juga melakukan operasi pemotongan pada string. Bentuk umum syntax ekspresi pemotongan string:

```
<string>[<awal>:<akhir>]
```

Syntax di atas menghasilkan sebuah string baru dengan karakter-karakter dimulai dari karakter pada indeks `<awal>` dari `<string>` sampai dengan karakter indeks `<akhir> - 1` dari `<string>`.

Contoh pemotongan string:

```
>>> nama_lengkap = 'Budi Susilo'
>>> nama_lengkap[0:4]
'Budi'
```

Jika kita tidak menuliskan `<awal>` pada ekspresi pemotongan string, maka indeks 0 akan digunakan sebagai `<awal>`:

```
>>> nama_lengkap = 'Budi Susilo'
>>> nama_lengkap[:4]
'Budi'
```

Jika kita tidak menuliskan `<akhir>` pada ekspresi pemotongan string, maka Panjang dari string akan digunakan sebagai `<akhir>`:

```
>>> nama_lengkap = 'Budi Susilo'
>>> nama_lengkap[5:]
'Susilo'
```

Jika kita tidak menuliskan `<awal>` dan `<akhir>` pada ekspresi pemotongan string, maka kita akan mendapatkan salinan keseluruhan string:

```
>>> nama_lengkap = 'Budi Susilo'
>>> nama_lengkap[:]
'Budi Susilo'
```

Ekspresi pemotongan string juga dapat ditambahkan step untuk mendapatkan potongan string yang melewati beberapa karakter:

```
>>> huruf = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> huruf[0:26:2]
'ACEGIKMOQSUWY'
```

4.5.3 Konkatenasi String

Operasi yang umum dilakukan pada string adalah konkatenasi. Konkatenasi berarti penyambungan. Kita mengkonkatenasi dua string dengan operator `+`:

```
>>> pesan = 'Hello ' + 'world!'
>>> print(pesan)
Hello world!
```

Contoh lain konkatenasi string:

```
>>> nama_depan = 'Budi'
>>> nama_belakang = 'Susilo'
>>> nama_lengkap = nama_depan + ' ' + nama_belakang
>>> print(nama_lengkap)
Budi Susilo
```

Kita juga dapat menggunakan operator augmented assignment untuk melakukan konkatenasi:

```
>>> huruf = 'abc'
>>> huruf += 'def'
>>> print(huruf)
abcdef
```

Kita dapat melipatgandakan string menggunakan operator `*`:

```
>>> str1 = 'Hello'
>>> str1 * 3
'HelloHelloHello'
```

4.5.4 String Bersifat *Immutable*

Immutable berarti tidak dapat berubah. String bersifat immutable berarti karakter-karakter dari string tidak dapat kita ubah. Jika kita mencoba mengubah karakter dalam string menggunakan pengindeksan kita akan mendapatkan error:

```
>>> teman = 'Dodi'
>>> teman[1] = 'i'
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    teman[1] = 'i'
TypeError: 'str' object does not support item assignment
```

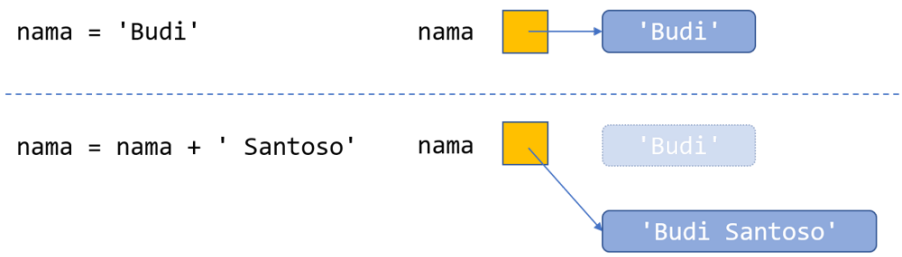
Beberapa operasi seperti konkatenasi terlihat seperti memodifikasi string, namun sebenarnya operasi-operasi tersebut tidak mengubah string. Perhatikan contoh berikut:

```
>>> nama = 'Budi'
>>> nama = nama + ' Santoso'
>>> print(nama)
Budi Santoso
```

Pada baris 2 sesi interaktif di atas:

```
nama = nama + ' Santoso'
```

terlihat seperti memodifikasi string original (string dalam variabel `nama` sebelumnya, yaitu `'Budi'`) yang seharusnya tidak bisa dilakukan karena sifat string yang immutable. Statement tersebut tidaklah memodifikasi string original, namun membuat string baru yang merupakan hasil konkatenasi string dalam variabel `nama` sebelumnya dengan string `' Santoso'`. Gambar berikut mengilustrasikan ini:



Ketika kita mengkonkatenasi string, interpreter tidak mengubah string original, namun membuat string baru hasil konkatenasi. String original masih tersimpan dalam memori.

4.5.4 Method-method pada String

Program yang memanipulasi string sangatlah umum. Hampir semua program akan melibatkan manipulasi string. Python menyediakan banyak method untuk memanipulasi string dan dapat dikelompokkan menurut operasi yang dilakukan:

- Method yang melakukan pengujian string
- Method yang melakukan modifikasi string
- Method yang melakukan pencarian dan penggantian karakter
- Method `split()`

Method-method yang Melakukan Pengujian String

Table di bawah mendaftar method-method string yang digunakan untuk melakukan pengujian terhadap string untuk suatu sifat tertentu:

Method	Keterangan
`<string>.isalnum()`	Mengembalikan `True` jika `<string>` hanya terdiri dari huruf-huruf alfabet atau digit-digit angka, dan setidaknya mempunyai panjang satu karakter. Mengembalikan `False` jika sebaliknya.
`<string>.isalpha()`	Mengembalikan `True` jika `<string>` hanya terdiri dari huruf-huruf alfabet, dan setidaknya mempunyai panjang satu karakter. Mengembalikan `False` jika sebaliknya.
`<string>.isdigit()`	Mengembalikan `True` jika `<string>` hanya terdiri dari digit-digit angka, dan setidaknya mempunyai panjang satu karakter. Mengembalikan `False` jika sebaliknya.
`<string>.islower()`	Mengembalikan `True` jika semua huruf-huruf alfabet dalam `<string>` berupa huruf kecil, dan setidaknya mempunyai panjang satu karakter. Mengembalikan `False` jika sebaliknya.
`<string>.isspace()`	Mengembalikan `True` jika `<string>` hanya terdiri dari karakter whitespace. Mengembalikan `False` jika sebaliknya. Karakter whitespace adalah spasi, baris baru (\n) dan tab (\t).
`<string>.isupper()`	Mengembalikan `True` jika semua huruf-huruf alfabet dalam `<string>` berupa huruf besar, dan setidaknya mempunyai panjang satu karakter. Mengembalikan `False` jika sebaliknya.

`isalnum` adalah singkatan dari "is alphanumeric" yang berarti “apakah alfanumerik (huruf dan angka) ?”. Method `isalnum()` menguji apakah karakter-karakter dalam string hanya terdiri dari karakter huruf dan angka. Contoh penggunaan method `isalnum()`:

```
>>> str1 = 'abc123'
>>> str1.isalnum()
True
>>> str2 = 'abc123@gmail.com'
>>> str2.isalnum()
False
```

`isalpha` adalah singkatan dari "is alphabet" yang berarti “apakah alfabet (huruf saja) ?”. Method `isalpha()` menguji apakah karakter-karaketer dalam string hanya terdiri dari karakter alfabet. Contoh penggunaan method `isalpha()`:

```
>>> str1 = 'abc123'
>>> str1.isalpha()
False
>>> str3 = 'abcdef'
>>> str3.isalpha()
True
```

`isdigit` berarti “apakah digit (angka) ?”. Method `isdigit()` menguji apakah karakter-karakter dalam string hanya terdiri dari karakter angka. Contoh penggunaan method `isdigit()`:

```
>>> str3 = 'abcdef123'
>>> str3.isdigit()
False
>>> str4 = '12345'
>>> str4.isdigit()
True
```

`islower` adalah singkatan "is lowercase" yang berarti “apakah lowercase (huruf kecil) ?”. Method `islower()` menguji apakah karakter-karakter dalam string hanya terdiri dari huruf kecil. Contoh penggunaan method `islower()`:

```
>>> str5 = 'abc123'
>>> str5.islower()
True
>>> str6 = 'abc123$@%&'
>>> str6.islower()
True
>>> str7 = 'Abcd123'
>>> str7.islower()
False
```

`isspace` berarti “apakah ada spasi?”. Method `isspace()` menguji apakah karakter-karakter dalam string hanya terdiri dari spasi. Contoh penggunaan method `isspace()`:

```
>>> str8 = 'selamat datang'
>>> str8.isspace()
False
>>> str9 = '          '
>>> str9.isspace()
True
```

`isupper` adalah singkatan dari "is uppercase" yang berarti “apakah uppercase (huruf besar)?”. Method `isupper()` menguji apakah karakter-karakter dalam string hanya terdiri dari huruf besar.Contoh penggunaan method `isupper()`:

```
>>> str10 = 'selamat datang'
>>> str10.isupper()
False
>>> str11 = 'SELAMAT DATANG'
>>> str11.isupper()
True
>>> str12 = 'Selamat Datang'
>>> str12.isupper()
False
```

Method-method yang Melakukan Modifikasi String

String mempunyai method-method yang dapat digunakan memodifikasi karakter dalam string. Karena string adalah immutable, method-method ini tidak mengubah langsung karakter string, namun mengembalikan salinan string hasil modifikasi. Misalkan, string mempunyai method `lower()` yang digunakan untuk mendapatkan salinan string yang semua karakter dalam string yang berupa huruf besar dikonversi ke huruf kecil.

```
>>> str1 = 'HELLO WORLD'
>>> str_lower = str1.lower()
>>> print(str_lower)
hello world
>>> print(str1)
HELLO WORLD
```

`str1.lower()` mengembalikan salinan string `str1` dengan semua karakter yang berupa huruf besar dikonversi ke huruf kecil. Pemanggilan method ini tidak mengubah string yang disimpan pada `str1`. Ini berbeda dengan method pada list yang mengubah isi list original.

Tabel berikut mendaftar method string yang mengembalikan hasil modifikasi karakter dalam string:

Method	Keterangan
`<string>.lower()`	Mengembalikan salinan dari `<string>` dengan semua karakter diubah ke huruf kecil.
`<string>.lstrip()`	Mengembalikan salinan dari `<string>` dengan semua karakter whitespace yang berada di awal string dihapus. Karakter whitespace adalah spasi, baris baru `(\n)`, dan `tab(\t)`.
`<string>.lstrip(char)`	Mengembalikan salinan dari `<string>` dengan semua karakter `char` yang berada di awal string dihapus.
`<string>.rstrip()`	Mengembalikan salinan dari `<string>` dengan semua karakter whitespace yang berada di akhir string dihapus. Karakter whitespace adalah spasi, baris baru `(\n)`, dan `tab(\t)`.
`<string>.rstrip(char)`	Mengembalikan salinan dari `<string>` dengan semua karakter `char` yang berada di akhir string dihapus.
`<string>.strip()`	Mengembalikan salinan dari `<string>` dengan semua karakter whitespace yang berada di dalam string dihapus.
`<string>.strip(char)`	Mengembalikan salinan dari `<string>` dengan semua karakter `char` yang berada di dalam string dihapus
`<string>.upper()`	Mengembalikan salinan dari `<string>` dengan semua karakter berupa huruf dikonversi ke huruf besar.
`<string>.capitalize()`	Mengembalikan salinan dari `<string>` dengan huruf pertama dikonversi ke huruf besar.

`lstrip` adalah singkatan left strip yang berarti lucuti (hilangkan) bagian kiri. Method `lstrip()` dapat digunakan dengan argumen maupun tanpa argumen.

Method `lstrip()` tanpa argumen menghapus spasi-spasi di bagian kiri string. Berikut contoh penggunaan method `lstrip()` tanpa argumen:

```
>>> str2 = '      selamat datang'
>>> str2.lstrip()
'selamat datang'
>>> str3 = '\t\n Selamat Datang'
>>> print(str3)

 Selamat Datang
>>> print(str3.lstrip())
Selamat Datang
```

Method `lstrip()` dengan argumen menghapus karakter-karakter sesuai dengan argumen di bagian kiri string (sampai dengan karakter yang berbeda dari argument). Argumen dari method ini dapat lebih dari satu karakter. Ketika digunakan dengan argumen lebih dari satu karakter, method `lstrip()` menghapus semua karakter-karakter di bagian kiri string yang sama dengan karakter-karakter dalam argumen sampai dengan karakter yang tidak ada dalam argumen. Berikut contoh penggunaan method `lstrip()` dengan argumen:

```
>>> str4 = 'xxxxSxxxxSxxxx'
>>> str4.lstrip('x')
'SxxxxSxxxx'
>>> web = 'http://www.example.com'
>>> web.lstrip('http:/.w')
'example.com'
```

`rstrip` adalah singkatan right strip yang berarti lucuti (hilangkan) bagian kanan. `rstrip()` dapat dipanggil tanpa argument maupun dengan argument. Method `rstrip()` bekerja mirip dengan method `lstrip()`, hanya saja `rstrip()` menghapus karakter-karakter yang sesuai dengan argumen (menghapus spasi jika dipanggil tanpa argumen) di bagian kanan string. Berikut contoh penggunaan method `rstrip()` tanpa argumen:

```
>>> str5 = 'selamat datang         '
>>> str5.rstrip()
'selamat datang'
```

Contoh penggunaan method `rstrip()` dengan argumen dapat dilihat pada sesi interaktif berikut:

```
>>> str4 = 'xxxxSxxxxSxxxx'
>>> str4.rstrip('x')
'xxxxSxxxxS'
>>> web = 'example.com/wow'
>>> web.rstrip('wo/')
'example.com'
```

`strip` berarti lucuti (hilangkan). Method `strip()` dapat digunakan tanpa argumen maupun dengan argumen. Method ini menghapus karakter-karakter yang sesuai dengan argumen (menghapus spasi jika dipanggil tanpa argumen) di bagian kiri dan bagian kanan string. Berikut adalah contoh penggunaan method `strip()` tanpa argumen:

```
>>> str7 = '      selamat datang      '
>>> str7.strip()
'selamat datang'
>>> str8 = '  \t selamat datang  \n  '
>>> print(str8)
     selamat datang

>>> print(str8.strip())
selamat datang
```

Berikut adalah contoh penggunaan method `strip()` dengan argumen:

```
>>> str4 = 'xxxxSxxxxSxxxx'
>>> str4.strip('x')
'SxxxxS'
>>> web = 'www.example.com'
>>> web.strip('cmowz.')
'example'
```

`upper` adalah singkatan dari uppercase yang berarti huruf besar. Method `upper()` mengubah semua karakter-karakter huruf kecil dalam string menjadi huruf besar. Berikut contoh penggunaan method `upper()`:

```
>>> str9 = 'selamat datang'
>>> str9.upper()
'SELAMAT DATANG'
```

`capitalize` berarti kapitalkan (huruf pertama adalah huruf besar). Method `capitalize()` mengkapitalkan string. Berikut contoh penggunaan method `capitalize()`:

```
>>> str9 = 'selamat datang'
>>> str9.capitalize()
'Selamat datang'
```

Method-method yang Melakukan Pencarian dan Penggantian String

Banyak program melakukan pencarian suatu karakter atau barisan karakter (substring) dari sebuah string. Misalkan, program word prosesor memiliki fitur yang melakukan pencarian sebuah kata dalam sebuah dokumen.

Tabel berikut mendaftar method-method string yang digunakan untuk pencarian substring dan penggantian substring dengan substring lain:

Method	Keterangan
`<string>.endswith(substring)`	Argumen `substring` adalah sebuah string. Method ini mengembalikan True jika `<string>` berakhir dengan `substring`.
`<string>.find(substring)`	Argumen `substring` adalah sebuah string. Method ini mengembalikan indeks terendah dalam string dimana `substring` ditemukan. jika `substring` tidak ditemukan maka method ini mengembalikan `-1`.
`<string>.replace(old, new)`	Argumen old dan new keduanya adalah string. Method ini mengembalikan salinan dari string dengan semua.
`<string>.startswith(substring)`	Argumen `substring` adalah sebuah string. Method ini mengembalikan True jika `<string>` dimulai dengan `substring`.

`endswith` berarti berakhir dengan. Method `endswith()` menguji apakah string berakhir dengan string yang diberikan sebagai argumen. Berikut contoh penggunaan method `endswith()`:

```
>>> nama_file = 'dokumen.txt'
>>> nama_file.endswith('.txt')
True
>>> nama_file.endswith('.py')
False
```

`find` berarti temukan. Method `find()` mencari apakah di dalam string terdapat string yang diberikan sebagai argumen. Method ini mengembalikan nomor indeks dari karakter pertama dari bagian string yang cocok. Berikut contoh penggunaan method `find()`:

```
>>> teks = 'Halo, selamat datang di dunia pemrograman'
>>> teks.find('selamat')
6
>>> teks.find('di')
21
>>> teks.find('program')
-1
```

replace berarti tukar. Method `replace()` menerima dua argumen. Argumen pertama berupa string yang ingin dicari dan argumen kedua adalah string yang digunakan untuk menggantikan string yang dicari jika ditemukan. Berikut contoh penggunaan method `replace()`:

```
>>> teks = 'Saya menyukai apel'
>>> teks.replace('apel', 'jeruk')
'Saya menyukai jeruk'
```

`startswith` berarti mulai dengan. Method `starswith()` menguji apakah string dimulai dengan string yang diberikan sebagai argumen. Berikut contoh penggunaan method `startswith()`:

```
>>> teks = 'Halo, selamat datang di dunia pemrograman'
>>> teks.startswith('Halo')
True
>>> teks.startswith('halo')
False
```

Method `split`

String juga mempunyai method `split()` yang mengembalikan sebuah list berisi potongan-potongan string berdasarkan suatu pemisah. Syntax pemanggilan method `split()`:

```
<str>.split(<pemisah>)
```

Argumen `<pemisah>` adalah opsional. Jika kita tidak memberikan `<pemisah>`, maka method `split()` mengembalikan sebuah list berisi potongan-potongan string yang dipisah berdasarkan spasi. Berikut adalah contoh program yang menggunakan method `split()` tanpa `<pemisah>`:

```
# string_split.py
# Program ini mendemonstrasikan method split
def main():
    # Buat sebuah string yang terdiri dari kata-kata
    my_string = 'satu dua tiga empat'

    # Split (pisahkan) string
    list_kata = my_string.split()

    # Tampilkan list dari kata-kata dalam string
    print(list_kata)

# Panggil fungsi main
main()
```

Output:

```
['satu', 'dua', 'tiga', 'empat']
```

Kita dapat menentukan pemisah berbeda dengan memberikannya sebagai argumen `<pemisah>` ke method `split()`. Sebagai contoh, misalkan, sebuah string berisi tanggal: `tanggal_string = '10/03/2021'`. Jika kita ingin memisahkan tanggal, bulan, dan tahun masing-masing sebagai elemen dalam sebuah list, kita dapat memanggil method `split()` dengan argument `'/'`, seperti terlihat pada statement berikut:

```
tanggal_list = tanggal_string('/')
```

Setelah statement di atas dieksekusi, variabel `tanggal_list` akan mereferensikan list berikut:

```
['10', '03', '2021']
```

Berikut adalah contoh program mendemonstrasikan method `split()` dengan argumen:

```
# split_tanggal.py
```

```
# Program ini memanggil method split, dan
# menggunakan karakter '/' sebagai pemisah
def main():
    # Buat sebuah string tanggal
    string_tanggal = '10/03/2021'

    # Pisahkan tanggal, bulan, tahun
    list_tanggal = string_tanggal.split('/')

    # Tampilkan tanggal, bulan, dan tahun
    print('Tanggal:', list_tanggal[0])
    print('Bulan:', list_tanggal[1])
    print('Tahun:', list_tanggal[2])

# Panggil fungsi main
main()
```

Output:

```
Tanggal: 10
Bulan: 03
Tahun: 2021
```

4.6 Memroses String

Dalam membuat program kita akan seringkali melakukan pemrosesan string. Pada bagian ini kita akan membahas dua contoh program yang melakukan pemrosesan string:

- Username Generator
- Mengkonversi angka bulan ke singkatan nama bulan

4.6.1 Username Generator

Banyak sistem komputer menggunakan kombinasi username dan password untuk mengautentikasi pengguna. Sistem administrator harus memberikan username yang unik untuk setiap pengguna. Seringkali, username diambil dari nama pengguna yang dikombinasikan dengan nomor identitas pengguna. Program berikut menggenerasi username berdasarkan nama pengguna dan nomor identitas pengguna:

```
def main():
    # Minta nama depan, nama belakang, dan ID user
    nama_depan = input('Masukkan nama depan Anda: ')
    nama_belakang = input('Masukkan nama belakang Anda: ')
    nomor_id = input('Masukkan nomor ID Anda: ')

    set1 = nama_depan[0 : 3]

    set2 = nama_belakang[0 : 3]

    set3 = nomor_id [-3 :]

    nama_login = set1 + set2 + set3

    print('Login Anda adalah: ', nama_login)

main()
```

Output:

```
Masukkan nama depan Anda: Rasya
Masukkan nama belakang Anda: Putra
Masukkan nomor ID Anda: 3
Login Anda adalah: RasPut3
```

4.6.2 Mengkonversi Angka Bulan ke Singkatan Nama Bulan

Berikut adalah contoh program yang mengkonversi angka bulan ke singkatan nama bulan :

```
#konversi angka bulan ke singkatan nama bulan
#konversi tanggal dari format "dd/mm/yyy" ke "hari bulan, tahun"
def main():
    #Masukkan tanggal
    tanggalstr = input('Masukkan tanggal (dd/mm/yyy): ')
    #split masing-masing komponen
    haristr, bulanstr, tahunstr = tanggalstr.split('/')
    #Konversi bulanstr ke nama bulan
    bulan =
['Jan','Feb','Mar','Apr','Mei','Jun','Jul','Ags','Sept','Okt','Nov','Des']
    bulanstr = bulan[int(bulanstr) - 1]

    print('Konversi tanggal adalah :',haristr,bulanstr+',',tahunstr)

main()
```

Output:

```
Masukkan tanggal (dd/mm/yyy): 11/03/2021
Konversi tanggal adalah : 11 Mar, 2021
```

4.6.3 Penvalidasi Password

Kita diminta mengembangkan program penggenerasi username untuk meminta pengguna menentukan password dari login mereka. Password harus mengikuti ketentuan berikut:

- Harus terdiri setidaknya tujuh karakter
- Harus terdiri dari setidaknya satu huruf besar
- Harus terdiri dari setidaknya satu huruf kecil
- Harus terdiri dari setidaknya satu digit angka

Kita memutuskan untuk membuat fungsi baru dengan nama `validasi_password` yang menvalidasi password yang dimasukkan pengguna dan memodifikasi fungsi main:

- Fungsi `validasi_password` menerima password dan mengembalikan True jika password mengikuti dan mengembalikan False jika password tidak mengikuti ketentuan
- Fungsi `main` meminta password dan mengulang permintaan password sampai pengguna memasukkan password yang mengikuti ketentuan

Berikut ini adalah contoh program penvalidasi password :

```
#program menggenerasi username
def generasi_username(nama_depan,nama_belakang,id):
    #ambil 3 huruf pertama nama depan
    set1 = nama_depan[0:3].lower()

    #ambil tiga huruf pertama nama_belakang
    set2 = nama_belakang[0:3].lower()

    #ambil empat karakter terakhir dari id
    set3 = id[-4:]

    #konkatenasi ketiga set
    username = set1 + set2 + set3

    #kembalikan username
    return username
def validasi_password(password):
    #variabel untuk menyimpan hasil pengujian ketentuan
    #inisialisasi dengan false
    panjang_benar = False
    ada_huruf_besar = False
    ada_huruf_kecil = False
    ada_digit = False

    #Mulai validasi dengan uji panjang karakter
    if len(password)>=7:
        panjang_benar=True

        #Uji setiap ketentuan dan tetapkan variabel ketentuan ke true
        for ch in password:
            if ch.isupper():
                ada_huruf_besar = True
            if ch.islower():
                ada_huruf_kecil = True
            if ch.isdigit():
                ada_digit = True

    #Jika semua ketentuan terpenuhi (True)
                #Tetapkan variabel is_valid dengan true
    if panjang_benar and ada_huruf_besar and \
       ada_huruf_kecil and ada_digit:
        is_valid = True
    else:
        is_valid = False
    return is_valid


def main():
    #minta pengguna memasukkan nama
    nama = input('Masukkan nama anda: ')
    #minta id pengguna
    id_mhs = input('Masukkan id mahasiswa anda: ')

    #pisahkan nama depan dan belakang
    nama_list = nama.split()
```

```
    #panggil fungsi generasi username dan simpan nilai ke sebuah variabel
    username = generasi_username(nama_list[0],nama_list[1],id_mhs)

    #Tampilkan username
    print(f'\nUsername Anda:{username}')

    #minta password
    password = input('Masukkan password baru anda:')

    #validasi password dan minta kembali jika password yang dimasukkan tidak
sesuai
    while not validasi_password(password):
        print('Password yang anda masukkan tidak valid')
        password = input('Masukkan password baru anda:')

        #Tampilkan password
        print('Password anda valid')

main()
```

Output :

```
Masukkan nama anda: Rasya Putra
Masukkan id mahasiswa anda: 15114156

Username Anda:rasput4156
Masukkan password baru anda:Abcd123
Password anda valid
```

REFERENSI:

[1] Gaddis, Tony. 2012. *Starting Out With Python Second Edition*. United States of America: Addison-Wesley.