MODUL VIII

FUNGSI *BUILT-IN* PADA PYTHON

A. Tujuan

Tujuan dari modul ini adalah sebagai berikut.

1. Mahasiswa mampu menggunakan fungsi-fungsi *string* pada Python
2. Mahasiswa mampu menggunakan fungsi-fungsi numerik pada Python
3. Mahasiswa mampu menggunakan fungsi-fungsi *dictionary* pada Python
4. Mahasiswa mampu menggunakan fungsi-fungsi *list* pada Python

B. Fungsi-Fungsi String

Pada subbab ini akan mempelajari dasar dasar penggunaan *string*. Dalam subbab ini akan membahas lebih jauh tentang penggunaan metode-metode yang telah disediakan. Metode adalah fungsi yang didefinisikan di dalam suatu kelas.

1. Metode *Capitalize*()

Metode ini digunakan untuk mengubah huruf pertama pada *string* menjadi huruf kapital. Penulisan *syntax* dari metode ini adalah: *str.capitalize*(), dimana str adalah nama variabel yang memuat data bertipe *string*. Contoh penggunaannya dapat dilihat pada contoh 8.1 berikut.

Contoh 8.1

```
str = "hello world"
s = str.capitalize()
print(s) #output: Hello world
```

Berdasarkan contoh 8.1, terlihat bahwa huruf h pada awal *string* yang awalnya huruf biasa kemudian berubah menjadi huruf kapital.

2. Metode *Center*()

Metode *center* digunakan untuk meletakkan suatu kalimat berada di tengah, dimana ruang kanan dan ruang kiri kalimat akan diisi dengan suatu karakter yang ditentukan. Penulisan *syntax* dari metode ini adalah: *str.center*(*width*, *fillchar*). Parameter *width* adalah lebar *string* yang diinginkan, sedangkan *fillchar* adalah karakter yang akan mengisi ruang kanan dan kiri Contoh penggunaan dapat dilihat pada contoh 8.2 berikut

Contoh 8.2

```
str = "Hello World"
s = str.center(20,'*')
print(s) #output: ****Hello World*****
s = str.center(19,'*')
print(s) #output: ****Hello World****
```

Pada saat kita mengetik kode s = str.center(20,'*'), *output* nya adalah tulisan "*Hello world*" berada di tengah, dimana ruang kanan dan kiri akan diisi dengan karakter '*'. Lebar *string* "*Hello world*" adalah 11, dimana ini merupakan jumlah karakter dan spasi pada *string* tersebut. Pada awalnya, parameter *width* diatur sama dengan 20, maka jumlah karakter '*" pada *output* yang mengisi ruang kanan dan kiri sama dengan (20 – 11)/2 = 4,5. Karena hasilnya adalah bilangan pecahan, karena bilangan bulat terdekat adalah 4 dan 5, maka jumlah karakter '*' pada salah satu sisi sama dengan lima, sedangkan sisi berlawanannya sama dengan empat. Pada contoh ini, hasil *output* nya adalah ruang kanan diisi 5 karakter '*' dan ruang kiri diisi 4 karakter '*' Berdasarkan hasil ini, dapat disimpulkan bahwa parameter *width* pada kode *str.center*(*width*, *fillchar*) tidak boleh kurang dari lebar *string* yang akan kita letakkan di tengah dan jika mengkehendaki jumlah karakter pada ruang kanan dan kiri adalah sama, maka hasil pembagian tidak boleh berupa bilangan pecahan

Selanjutnya adalah mengganti parameter *width* dari 20 menjadi 19, sehingga kodenya adalah s = str.center(19,'*'). *Output* nya sama seperti sebelumnya, hanya saja jumlah jumlah karakter '*" pada *output* yang mengisi ruang kanan dan kiri *string* "*Hello world*" berubah menjadi sama dengan (19 – 11)/2 = 4. Karena hasilnya adalah bilangan bulat, maka jumlah karakter '*" pada ruang kiri dan kanan adalah sama.

3. Metode *Count*()

Metode ini berfungsi untuk menghitung berapa kali *substring* muncul di dalam *sring* str, maupun dapat di-*set* dimulai dari indeks ke-m dan berakhir di indeks ke-n. Penulisan *syntax* dari metode ini adalah: *str.count*(*substr, m, n*). Parameter kedua dan ketiga bersifat opsional, jika hanya ditulis *str.count*(*substr*), maka program akan membacanya sebagai menghitung jumlah *substring* secara

keseluruhan dalam *string* str. Contoh penggunaan dapat dilihat pada Contoh 8.3 berikut.

Contoh 8.3

```
str = "Hello World"
s = str.count('l')
print(s) #output: 3
s = str.count('o')
print(s) #output: 2
s = str.count('lo')
print(s) #output: 1
s = str.count('l', 0, 3) #menghitung jumlah substring 'l' antara indeks ke-0 hingga indeks ke-3
print(s) #output: 1
s = str.count('l', 0, 4) #menghitung jumlah substring 'l' antara indeks ke-0 hingga indeks ke-4
print(s) #output: 2
s = str.count('l', 2, 4) #menghitung jumlah substring 'l' antara indeks ke-2 hingga indeks ke-4
print(s) #output: 2
s = str.count('l', 0, len(str)) #menghitung jumlah substring 'l' dari indeks awal (indeks ke-0) hingga akhir
s = str.count('H')
print(s) #output: 1
s = str.count('h')
print(s) #output: 0
```

Pada saat nilai variabel s di-*set* sama dengan *str.count*(‘l’), maka variabel s akan bernilai sama dengan jumlah karakter ‘l’ pada kalimat “*Hello World*”, yaitu sama dengan 3. Pada saat nilai variabel s di-*set* sama dengan *str.count*(‘l’), maka variabel s akan bernilai sama dengan jumlah karakter ‘lo’ pada kalimat “*Hello World*”, yaitu sama dengan 1. Pada saat nilai variabel s di-*set* sama dengan *str.count*(‘l’, 0, 3), maka variabel s akan bernilai sama dengan jumlah karakter ‘l’ pada kalimat “*Hello World*” dari indeks ke-0 sampai indeks ke-2 saja (perlu diingat kembali, bahwa penomoran indeks pada program Python dimulai dari 0). Karakter pada indeks ke-3 selaku indeks *end* tidak ikut terbaca, sehingga program hanya akan menghitung karakter ‘l’ pada bagian “Hel” saja. Nomor indeks karakter ‘l’ kedua pada *substring* ‘*Hello’* adalah 3. Sehingga jika menetapkan indeks akhir pada kode *str.count* sama dengan 3, maka karakter ‘l’ yang terhitung hanya 1 saja. Agar karakter ‘l’ kedua pada *substring* ‘*Hello’* yang bernomor indeks sama dengan 3 masuk dalam perhitungan, maka kodenya harus diubah menjadi *str.count*(‘l’, 0, 4), sehingga hasil *output* nya adalah 2. Pada saat nilai variabel s di-*set* sama dengan *str.count*(‘l’, 2, 4), maka variabel s akan bernilai sama dengan jumlah karakter ‘l’ pada kalimat “*Hello World*” dari indeks ke-2 sampai indeks ke-3 saja. Karakter ‘l’ pertama pada kalimat “*Hello World*” bernomor indeks ke-2. Dengan demikian, saat nilai variabel s dicetak, maka *output* nya sama dengan 2, karena karakter ‘l’ pertama yang terletak pada indeks ke-2 juga dihitung, begitu juga dengan karakter ‘l’ kedua yang terletak pada indeks ke-3.

Pada saat nilai variabel s di-*set* sama dengan *str.count*(‘H’), maka variabel s akan bernilai sama dengan jumlah karakter ‘H’ pada kalimat “*Hello World*”, yaitu sama dengan 1, sedangkan pada saat nilai variabel s di-*set* sama dengan *str.count*(‘h’), maka variabel s akan bernilai sama dengan jumlah karakter ‘h’ pada kalimat “*Hello World*”, yaitu sama dengan 0. Mengapa bisa seperti ini? Perlu diingat kembali bahwa pemrograman Python bersifat *case sensitive* (Membedakan antara hruuf kapital dengan huruf biasa). Karena pada kalimat “*Hello World*” tidak ada karakter ‘h’, adanya adalah karakter ‘H’, maka pada saat variabel s = *str.count*(‘h’) dicetak, maka akan menghasilkan *output* sama dengan 0.

4. Metode *Endswith*()

Metode ini berguna untuk memeriksa apakah *string* str diakhiri oleh substr atau tidak, maupun dapat di-*set* untuk memeriksa apakah *string* str dari indeks ke-m sampai ke-n diakhiri oleh substr atau tidak. Jika benar maka akan mengembalikan nilai *True*. Namun jika salah, akan mengembalikan nilai *False*. Penulisan *syntax* dari metode ini adalah *str.endswith*(*substr, m, n*). Parameter kedua dan ketiga bersifat opsional, jika hanya ditulis *str. endswith*(*substr*), maka program akan membacanya sebagai memeriksa apakah *string* str (secara keseluruhan) diakhiri oleh *substr* atau tidak. Contoh penggunaan dapat dilihat pada Contoh 8.4 berikut.

Contoh 8.4

```
str = "Hello World"
s = str.endswith('World')
print(s) #output: True
s = str.endswith('world')
print(s) #output: False
s = str.endswith('lo', 0, 4) #memeriksa string str dari indeks ke-0 sampai ke-4 diakhiri oleh substr 'll' atau tidak
print(s) #output: False
s = str.endswith('lo', 0, 5) #memeriksa string str dari indeks ke-0 sampai ke-5 diakhiri oleh substr 'll' atau tidak
print(s) #output: True
```

Pada saat nilai variabel s di-*set* sama dengan *str.endswith*(‘*World*’), maka variabel s akan bernilai *True* karena *string* str “*Hello World*” memang benar diakhiri dengan *substring* “*World*”. Pada saat nilai variabel s di-*set* sama dengan *str.endswith*(‘*world*’), maka variabel s akan bernilai *False* karena *string* str “*Hello World*” bukan diakhiri dengan *substring* “*world*” melainkan “*World*”, perbedaan pada huruf kapital berpengaruh terhadap *output*. Pada saat nilai variabel s di-*set* sama dengan *str.endswith*(‘lo’, 0, 4), maka program hanya memeriksa *substring* “*Hello World*” dari indeks ke-0 sampai indeks ke-3 saja. Karakter pada indeks ke-4 selaku indeks *end* tidak akan diikutsertakan, sehingga program hanya akan

memeriksa pada bagian “Hell” saja. Nomor indeks karakter ‘o’ pada *substring* ‘*Hello’* adalah 4. Sehingga jika menetapkan indeks *end* pada kode *str.endswith* sama dengan 4, maka *str.endswith*(‘lo’, 0, 4) akan bernilai *False* karena “Hell” tidak diakhiri dengan *substring* “lo”. Agar karakter ‘o’ pada *substring* ‘*Hello’* yang bernomor indeks sama dengan 4 diikutsertakan dalam pemeriksaan, maka kodenya harus diubah menjadi *str.endswith*(‘lo’, 0, 5), sehingga hasil *output* nya adalah *True*.

5. Metode *Find*()

Metode ini digunakan untuk mencari substr di dalam *string* str. Jika ditemukan, metode akan mengembalikan nilai indeks atau posisi dari substirng bersangkutan. Jika tidak ditemukan, nilai yang akan dikembalikan adalah -1. Penulisan *syntax* dari metode ini adalah *str.find*(*substr, m, n*), dimana m adalah nomor indeks *start* dan n adalah nomor indeks *end*. Jika menggunakan kode ini, maka pencarian dilakukan dimulai dari indeks ke-m hingga indeks ke-n, namun karakter pada indeks ke-n tidak ikut terbaca. Parameter indeks *start* dan indeks *end* sifatnya opsional, sehingga boleh tidak ditulis. Jika hanya ditulis *str.find*(*substr*), maka program akan membaca *string* secara keseluruhan, tidak hanya terbatas pada *range* indeks tertentu saja. Contoh penggunaan dapat dilihat pada Contoh 8.5 berikut.

Contoh 8.5

```
str = "Hello World"
s = str.find('World')
print(s) #output: 6
s = str.find('world')
print(s) #output: -1
s = str.find('o', 0, 4) #mencari substring 'o' pada string str dari indeks ke-0 sampai ke-4
print(s) #output: -1
s = str.find('o', 0, 5) #mencari substring 'o' pada string str dari indeks ke-0 sampai ke-5
print(s) #output: 4
```

Pada saat nilai variabel s di-*set* sama dengan *str.find*(‘*World*’), maka variabel s akan bernilai 6 karena nomor indeks huruf ‘W’ yang merupakan karakter pertama pada *substring* “*World*” sama dengan 6. Pada saat nilai variabel s di-*set* sama dengan *str.find*(‘*world*’), maka variabel s akan bernilai -1 karena program tidak dapat menemukan *substring* “world”, perbedaan pada huruf kapital berpengaruh terhadap *output*. Pada saat nilai variabel s di-*set* sama dengan *str.find*(‘o’, 0, 4), maka program hanya memeriksa *substring* “*Hello World*” dari indeks ke-0 sampai

indeks ke-3 saja. Karakter pada indeks ke-4 selaku indeks *end* tidak akan diikutsertakan, sehingga program hanya akan memeriksa pada bagian "Hell" saja. Nomor indeks karakter 'o' pada *substring* '*Hello'* adalah 4. Sehingga jika menetapkan indeks akhir pada kode *str.endswith* sama dengan 4, maka *str.find*('o', 0, 4) akan bernilai -1 karena tidak ditemukan *substring* 'o' pada 'Hell'. Agar karakter 'o' pada *substring* '*Hello'* yang bernomor indeks sama dengan 4 diikutsertakan dalam pencarian, maka kodenya harus diubah menjadi *str.find*('o', 0, 5), sehingga hasil *output* nya adalah 4.

6. Metode *Index*()

Metode ini fungsinya sama dengan *find*(), akan tetapi *index*() akan membangkitkan eksepsi (*error*) jika *substring* substr yang dicari tidak ditemukan. Penulisan *syntax* dari metode ini adalah *str.index*(*substr*, *m*, *n*), dimana m adalah nomor indeks *start* dan n adalah nomor indeks *end*. Jika menggunakan kode ini, maka pencarian dilakukan dimulai dari indeks ke-m hingga indeks ke-n, namun karakter pada indeks ke-n tidak ikut terbaca. Parameter indeks *start* dan indeks *end* sifatnya opsional, sehingga boleh tidak ditulis. Jika hanya ditulis *str.index*(*substr*), maka program akan membaca *string* secara keseluruhan, tidak hanya terbatas pada *range* indeks tertentu saja. Contoh penggunaan dapat dilihat pada Contoh 8.6 berikut.

Contoh 8.6

```
str = "Hello World"
s = str.index('World')
print(s) #output: 6
s = str.index('world')
print(s) #output: Error
s = str.index('o', 0, 4) #mengembalikan nilai indeks substring 'o' pada string str dari indeks ke-0 sampai ke-4
print(s) #output: Error
s = str.index('o', 0, 5) #mengembalikan nilai indeks substring 'o' pada string str dari indeks ke-0 sampai ke-5
print(s) #output: 4
```

7. Metode *Replace*()

Metode ini digunakan untuk mengganti *substring* pada *string* str menjadi *substring* baru sesuai dengan yang ditentukan. Penulisan *syntax* dari metode ini adalah *str.replace*(*old*, *new*, *count*). Parameter ketiga menunjukkan berapa jumlah maksimal *substring* yang akan di-*replace*, jika hanya ditulis *str.replace*(*old*, *new*), maka program akan melakukan *replace* pada *substring old* menjadi *substring new* secara keseluruhan. Contoh penggunaan dapat dilihat pada contoh 8.7 berikut.

Contoh 8.7

```
str = "Hello World"
replace1 = str.replace('Hello', 'Hai')
print(replace1) #output: Hai World
replace1 = replace1.replace('World', 'Dunia')
print(replace1) #output: Hai Dunia
replace2 = str.replace('l', 'i')
print(replace2) #output: Heiio Worid
replace2 = str.replace('l', 'i', 1)
print(replace2) #output: Heilo World
```

Perhatikan contoh 8.7, pada saat variabel replace1 di-*set* sama dengan *str.replace*(‘*Hello*’, ‘Hai’), maka variabel replace1 akan bernilai “Hai *World*”. Kekurangan metode *replace* adalah tidak bisa melakukan lebih dari satu penggantian dalam waktu bersamaan. Lalu bagaimana caranya jika ingin melakukan penggantian sebanyak lebih dari satu? Caranya adalah variabel replace1 di-*set* sama dengan *replace1.replace*(*old, new*), sehingga nantinya program akan mengganti *substring* lama pada *string* replace1. Dengan menulis kode *replace1.replace*(‘*World*’, ‘Dunia’), maka *string* “Hai *World*” akan menjadi “Hai Dunia”. Jika ingin melakukan penggantian lagi, maka di bawahnya tulis kembali kode *replace1 = replace1.replace(old, new)*.

Kode selanjutnya adalah variabel replace2 di-*set* sama dengan *str.replace*(‘l’, ‘i’), maka program akan mengganti *substring* ‘l’ pada *string* str “*Hello World*” menjadi ‘i’, sehingga variabel replace2 akan bernilai “Heiio Worid”. Lalu bagaimana jika ingin hanya satu *substring* ‘l’ saja yang diganti? Caranya dalah variabel replace2 di-set sama dengan *str.replace*(‘l’, ‘i’, 1), sehingga nantinya hanya ada 1 *substring* ‘l’ saja yang diganti, sehingga variabel replace2 akan bernilai “Heilo World”.

8. Metode *Upper*() dan *Lower*()

Metode ini digunakan untuk mengubah huruf dalam string str menjadi huruf besar maupun kecil. Metode *upper*() digunakan untuk huruf besar sedangkan metode lowerdigunakan untuk huruf kecil. Bentuk *syntax* dari metode ini adalah *str.upper*() dan *str.lower*(). Contoh penggunaan dapat dilihat pada contoh 8.8 berikut.

Contoh 8.8

```
str = "Hello World"
s = str.upper()
print(s) #output: HELLO WORLD
s = str.lower()
print(s) #output: hello world
```

C. Fungsi-Fungsi Numerik

Selain fungsi *string*, fungsi numerik juga umum dijumpai dalam kasus nyata suatu pemrograman. Fungsi-fungsi numerik digunakan untuk melakukan operasi-operasi tertentu terhadap suatu bilangan. Subbab ini akan membahas mengenai fungsi-fungsi numerik dalam pemrograman Python.

1. Fungsi abs()

Fungsi ini akan mengembalikan nilai mutlak dari suatu bilangan. Bentuk *syntax* dari fungsi ini adalah *abs*(*num*). Contoh penggunaannya dapat dilihat pada contoh 8.9 berikut.

Contoh 8.9

```
a = -0.71
b = 0.86
print('a =', a) #output: a = -0.71
print('b =', b) #output: b = 0.86
print('|a| =', abs(a)) #output: |a| = 0.71
print('|b| =', abs(b)) #output: |b| = 0.86
```

2. Fungsi ceil()

Fungsi ini akan mengembalikan nilai pembulatan ke atas dari suatu bilangan riil. Untuk menggunakan fungsi ini, maka perlu untuk memastikan bahwa kode *import math* sudah ada dalam program agar fungsi ini dapat digunakan. Bentuk *syntax* dari fungsi ini adalah *math.ceil*(*num*). Contoh penggunaannya dapat dilihat pada contoh 8.10 berikut.

Contoh 8.10

```
import math
a = 3.14
b = 2.71
print('[a] =', math.ceil(a)) #output: [a] = 4
print('[b] =', math.ceil(b)) #output: [b] = 3
```

3. Fungsi fabs()

Fungsi ini akan mengembalikan nilai mutlak dari suatu bilangan, sama seperti fungsi abs(). Untuk menggunakan fungsi ini, maka perlu untuk memastikan bahwa kode *import math* sudah ada dalam program agar fungsi ini dapat digunakan. Bentuk *syntax* dari fungsi ini adalah *math.fabs*(*num*). Contoh penggunaannya dapat dilihat pada contoh 8.11 berikut.

Contoh 8.11

```
import math
a = -0.71
b = 0.86
print('a =', a) #output: a = -10
print('b =', b) #output: b = 20
print('|a| =', math.fabs(a)) #output: |a| = 0.71
print('|b| =', math.fabs(b)) #output: |b| = 0.86
```

4. Fungsi floor()

Fungsi ini akan mengembalikan nilai pembulatan ke bawah dari suatu bilangan riil. Untuk menggunakan fungsi ini, maka perlu untuk memastikan bahwa kode *import math* sudah ada dalam program agar fungsi ini dapat digunakan. Bentuk *syntax* dari fungsi ini adalah *math.floor*(*num*). Contoh penggunaannya dapat dilihat pada contoh 8.12 berikut.

Contoh 8.12

```
import math
a = 3.14
b = 2.71
print('[a] =', math.floor(a)) #output: [a] = 3
print('[b] =', math.floor(b)) #output: [b] = 2
```

5. Fungsi min() dan max()

Fungsi ini digunakan untuk mencari nilai maksimum maupun minimum dari suatu data angka. Bentuk *syntax* dari fungsi ini adalah *max*(*num1*, *num2*, …) dan *min*(*num1*, *num2*,…), dimana *num1*, *num2*, dan seterusnya adalah suatu data angka (baik yang termuat dalam variabel maupun yang tidak termuat dalam variabel). Contoh penggunaannya dapat dilihat pada contoh 8.13 berikut.

Contoh 8.13

```
a = 3
b = 4
c = 5
print('Nilai max =', max(a,b,c)) #output: Nilai max = 5
print('Nilai min =', min(a,b,c)) #output: Nilai min = 3
```

6. Fungsi sqrt()

Fungsi ini digunakan untuk menghitung nilai akar kuadrat dari suatu data angka. Untuk menggunakan fungsi ini, maka perlu untuk memastikan bahwa kode *import math* sudah ada dalam program agar fungsi ini dapat digunakan. Bentuk *syntax* dari fungsi ini adalah *math.sqrt*(*num*). Contoh penggunaannya dapat dilihat pada contoh 8.14 berikut.

Contoh 8.14

```
import math
a = 9
b = 4
print('sqrt(a) =', math.sqrt(a)) #output: sqrt(a) = 3.0
print('sqrt(b) =', math.sqrt(b)) #output: sqrt(b) = 2.0
```

7. Fungsi choice()

Fungsi ini digunakan untuk mengambil nilai acak dari suatu objek, *list*, maupun *string*. Setiap proses pemanggilan fungsi akan mengembalikan nilai yang berbeda. Fungsi ini tersimpan dalam modul random. Oleh karena itu, maka perlu untuk memastikan bahwa kode *import random* sudah ada dalam program agar fungsi ini dapat digunakan. Bentuk *syntax* dari fungsi ini adalah *random.choice*(). Contoh penggunaannya dapat dilihat pada contoh 8.15 berikut.

Contoh 8.15

```
import random
list = [3,4,5,6,8,10]
print('random 1 choice =', random.choice(list)) #fungsi akan mengambil satu data acak pada list
print('random 2 choice =', random.choice(list))
print('random 3 choice =', random.choice(list))
print('Random choice one character from string "List":')
print(random.choice("List")) #fungsi akan mengambil satu karakter acak pada string
```

D. Fungsi-Fungsi *Dictionary*

Python menyediakan fungsi-fungsi yang dapat diterapkan untuk melakukan operasi operasiterhadap suatu *dictionary*. Pada subbab ini akan dibahas mengenai fungsi-fungsi *dictionary* pada Python.

1. Fungsi len()

Fungsi ini digunakan untuk mengembalikan jumlah elemen di dalam *dictionary*. Bentuk *syntax* dari fungsi ini adalah *len*(*dict*). Contoh penggunaannya dapat dilihat pada contoh 8.16 berikut.

Contoh 8.16

```
a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
b = {'satu' : 10, 'dua' : 20}
print('a =', a) #output: a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
print('b =', b) #output: b = {'satu' : 10, 'dua' : 20}
print('len a =', len(a)) #output: 3. Karena dalam dict a terdapat 3 elemen
print('len b =', len(b)) #output: 2. Karena dalam dict b terdapat 2 elemen
```

2. Metode clear()

Metode ini digunakan untuk mengosongkan atau menghapus semua elemen di dalam *dictionary*. Bentuk *syntax* dari metode ini adalah *dict.clear*(). Contoh penggunaannya dapat dilihat pada contoh 8.17 berikut.

Contoh 8.17

```
a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
b = {'satu' : 10, 'dua' : 20}
print('a =', a) #output: a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
print('b =', b) #output: b = {'satu' : 10, 'dua' : 20}
print('len a =', len(a)) #output: 3. Karena dalam dict a terdapat 3 elemen
print('len b =', len(b)) #output: 2. Karena dalam dict b terdapat 2 elemen
#menghapus elemen
a.clear()
b.clear()
#setelah dihapus
print('len a =', len(a)) #output: 0. Karena dalam dict a sudah tidak ada elemen
print('len b =', len(b)) #output: 0. Karena dalam dict b sudah tidak ada elemen
```

3. Metode copy()

Metode ini digunakan untuk membuat objek baru yang merupakan salinan dari *dictionary*. Bentuk *syntax* dari metode ini adalah *dict.copy*(). Contoh penggunaannya dapat dilihat pada contoh 8.18 berikut.

Contoh 8.18

```
a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
print('a =', a) #output: a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
#membuat copy a
b = a.copy()
print('b =', b) #output: a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
```

4. Metode items(), keys() dan values()

Metode *items* digunakan untuk mengembalikan daftar pasangan kunci dan nilai yang terdapat pada suatu *dictionary*. Metode *keys* digunakan untuk mendapatkan daftar kuncinya saja, sedangkan metode *values* digunakan untuk mendapatkan daftar nilainya saja. Bentuk *syntax* dari metode-metode ini adalah *dict*.*items*(), *dict*.*keys*() dan *dict*.*values*(). Contoh penggunaannya dapat dilihat pada contoh 8.19 berikut.

Contoh 8.19

```
a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
print('a =', a) #output: a = {'satu' : 10, 'dua' : 20, 'tiga' : 30}
item = a.items()
key = a.keys()
value = a.values()
print('item =', item) #output: item = dict_items([('satu', 10), ('dua', 20), ('tiga', 30)])
print('key =', key) #output: key = dict_keys(['satu', 'dua', 'tiga'])
print('value =', value) #output: value = dict_values([10, 20, 30])
```

E. Fungsi-Fungsi *List* Pada Python

Subbab ini akan membahas tentang fungsi-fungsi yang dapatditerapkan pada objek *list*.

1. Fungsi len()

Fungsi ini digunakan untuk mengembalikan jumlah elemen di dalam *list*. Bentuk *syntax* dari fungsi ini adalah *len*(*list*). Contoh penggunaannya dapat dilihat pada contoh 8.20 berikut.

Contoh 8.20

```
a = [3,4,5]
print('a =', a) #output: a = [3,4,5]
print('len a = ', len(a)) #output: len a = 3
```

2. Fungsi min() dan max()

Fungsi ini digunakan untuk mengembalikan nilai maksimum maupun minimum di dalam *list*. Elemen-elemen dalam *list* harus semuanya angka atau semuanya *string* (tidak boleh campuran angka dan *string*). Jika elemen dalam *list*

semuanya angka, maka fungsi akan mengembalikan angka terkecil atau terbesara, tergantung fungsi yang dipakai (*min* atau *max*). Jika elemen dalam *list* semuanya *string*, maka fungsi akan mencari *string* dalam *list* dengan urutan dimulai dari 0-9 lalu dilanjut dengan A-Z lalu dilanjut dengan a-z, kemudian mengembalikan nilai *string* dengan nomor urut terakhir jika fungsi yang digunakan adalah fungsi *max* dan akan mengembalikan nilai *string* dengan urutan pertama jika fungsi yang digunakan adalah fungsi *min*. Bentuk *syntax* dari dua fungsi ini adalah *list.max*() dan *list.min*(). Contoh penggunaannya dapat dilihat pada contoh 8.21 berikut.

Contoh 8.21

```
a = [0.71, 0.87, 1]
print('max a =', max(a)) #output: max a = 1
print('min a =', min(a)) #output: min a = 0.71
b = ['2021', 'bahasa', 'Python']
print('max b =', max(b)) #output: max b = bahasa
print('min b =', min(b)) #output: min b = 2021
c = ['Bahasa', 'Python']
print('max c =', max(c)) #output: max c = Python
print('min c =', min(c)) #output: min c = Bahasa
```

3. Metode append()

Metode ini digunakan untuk menambahkan suatu objek ke dalam *list* pada posisi terakhir. Bentuk *syntax* dari metode ini adalah *list.append*(*obj*). Contoh penggunaannya dapat dilihat pada contoh 8.22 berikut.

Contoh 8.22

```
a = ['Aku','suka']
print('a =', a) #output: a = ['Aku','suka']
a.append('pemrograman')
a.append('Python')
print('a setelah ditambah = ', a) #output: a = ['Aku', 'suka', 'pemrograman', 'Python']
```

Pada contoh 8.22, dapat dilihat bahwa objek ‘pemrograman’ dan ‘Python’ ditambahkan ke dalam *list* pada posisi setelah objek ‘suka’. Metode ini tidak dapat digunakan untuk menambahkan lebih dari satu objek secara sekaligus. Oleh karena itu, jika ingin menambahkan lebih dari satu objek, maka perlu untuk menulis kembali *syntax*.

4. Metode count()

Metode ini digunakan untuk menghitung jumlah atau banyaknya objek yang ada di dalam *list*. Bentuk *syntax* dari metode ini adalah *list.count*(*obj*). Contoh penggunaannya dapat dilihat pada contoh 8.23 berikut.

Contoh 8.23

```
a = [2, 0, 2, 2]
print('a =', a) #output: a = [2, 0, 2, 2]
print('jumlah nilai 2 =', a.count(2)) #output: jumlah nilai 2 = 3
print('jumlah nilai 0 =', a.count(0)) #output: jumlah nilai 0 = 1
```

5. Metode index()

Metode ini digunakan untuk mengetahui indeks dari suatu objek dalam *list*. Jika objek muncul beberapa kali, maka yang akan dikembalikan adalah indeks terendah. Bentuk *syntax* dari metode ini adalah *list.index*(*obj*). Contoh penggunaannya dapat dilihat pada contoh 8.24 berikut.

Contoh 8.24

```
a = [9, 9, 1, 4, 1, 6, 2, 7]
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6, 2, 7]
print('index (1) =', a.index(1)) #output: index (1) = 2
print('index (6) =', a.index(6)) #output: index (6) = 5
```

6. Metode insert()

Metode ini digunakan untuk memasukkan objek ke dalam *list* sebelum objek pada indeks tertentu. Metode ini sama seperti metode *append*(), hanya saja bedanya adalah pada metode *insert*(), objek dapat ditambahkan di posisi mana saja. Bentuk *syntax* dari metode ini adalah *list.insert*(*index*, *obj*). Contoh penggunaannya dapat dilihat pada contoh 8.25 berikut.

Contoh 8.25

```
a = ['pemrograman']
print('a =', a) #output: a = ['pemrograman']
#menambahkan objek 'Python' di posisi paling akhir
a.insert(len(a), 'Python')
print('a =', a) #output: a = ['pemrograman', 'Python']
#menambahkan objek 'Aku' di posisi paling awal
a.insert(0, 'Aku')
print('a =', a) #output: a = ['Aku', 'pemrograman', 'Python']
#menambahkan objek 'suka' di posisi sebelum 'pemrograman'
a.insert(1, 'suka')
print('a =', a) #output: a = ['Aku', 'suka', 'pemrograman', 'Python']
```

Sama halnya dengan metode *append*(), metode *insert*() tidak bisa digunakan untuk menambahkan dua objek atau lebih secara bersamaan. Jika ingin menambahkan dua objek, maka *syntax* harus ditulis kembali di bawahnya.

7. Metode pop()

Metode ini digunakan untuk menghapus objek yang berada pada list pada indeks tertentu. Bentuk *syntax* dari metode ini adalah *list.pop*(*index*). Parameter *index* dalam *syntax* ini merupakan nomor indeks dari objek yang akan dihapus. Parameter *index* sifatnya boleh untuk tidak ditulis. Apabila *syntax* hanya ditulis sebagai *list.pop*() saja, maka program akan menghapus objek pada indeks paling akhir. Apabila metode ini digunakan pada *list* kosong atau jika parameter *index* yang ditetapkan melebihi batas *range* indeks pada *list*, maka hasilnya akan *error* Contoh penggunaannya dapat dilihat pada contoh 8.26 berikut.

Contoh 8.26

```
a = [9, 9, 1, 4, 1, 6, 2, 7]
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6, 2, 7]
#menghapus 7 dari list
a.pop()
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6, 2]
#menghapus 2 dari list
a.pop()
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6]
#menghapus 4 dari list
a.pop(3)
print('a =', a) #output: a = [9, 9, 1, 1, 6]
```

8. Metode remove()

Metode ini digunakan untuk menghapus objek yang berada pada lisr objek. Jika objek lebih dari satu, maka yang akan dihapus adalah yang terletak pada indeks terendah. Bentuk *syntax* dari metode ini adalah *list.remove*(*obj*). Contoh penggunaannya dapat dilihat pada contoh 8.27 berikut.

Contoh 8.27

```
a = [9, 9, 1, 4, 1, 6, 2, 7]
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6, 2, 7]
#menghapus 7 dari list
a.remove(7)
print('a =', a) #output: a = [9, 9, 1, 4, 1, 6, 2]
#menghapus 6 dari list
a.remove(6)
print('a =', a) #output: a = [9, 9, 1, 4, 1, 2]
#menghapus 1 dari list
a.remove(1)
print('a =', a) #output: a = [9, 9, 4, 1, 2]
```

9. Metode reverse()

Metode ini digunakan untuk membalik urutan objek di dalam *list*. Bentuk *syntax* dari metode ini adalah *list.reverse*(). Contoh penggunaannya dapat dilihat pada contoh 8.28 berikut.

Contoh 8.28

```
a = [20, 21, 20, 22]
print('a =', a) #output: a = [20, 21, 20, 22]
a.reverse()
print('a setelah reverse =', a) #output: a = [22, 20, 21, 20]
```

10. Metode sort()

Metode ini digunakan untuk mengurutkan objek di dalam *list*. Jika ingin mengurutkan objek dalam *list* tanpa mengubah urutan aslinya, maka gunakan fungsi *sorted*(). *List* yang akan diurutkan harus berisi semua data angka atau semua data *string*, jika tidak maka akan *error*. Bentuk *syntax* dari metode ini adalah *list.sort*(*key*=… , *reverse*=…) dan *sorted*(*list*, *key*=… , *reverse*=…). Nilai parameter *key* berupa fungsi yang akan dijadikan acuan dalam urutan, sedangkan nilai parameter *reverse* berupa *True* atau *False*. Jika parameter *reverse* bernilai *False*, maka program akan mengurutkan *list* secara *ascending* (terkecil ke terbesar atau dari 0-9 ke A-Z ke a-z), jika bernilai *True*, maka program akan mengurutkan *list* secara *descending* (urutan sebaliknya). Dua parameter ini sifatnya boleh tidak ditulis, jika penulisan *syntax* hanya sebatas *list.sort*() dan *sorted*(*list*) saja, maka program akan secara *default* mengurutkan *list* secara *ascending* dan tidak berpatok pada fungsi apapun. Contoh penggunaannya dapat dilihat pada contoh 8.29 berikut.

Contoh 8.29

```
a = ['a', 'i', 'u', 'e', 'o']
print('a =', a) #output: a = ['a', 'i', 'u', 'e', 'o']
#mengurutkan objek dalam list a secara ascending
a.sort()
print('a =', a) #output: a = ['a', 'e', 'i', 'o', 'u']
#mengurutkan objek dalam list a secara descending
a.sort(reverse=True)
print('a =', a) #output: a = ['u', 'o', 'i', 'e', 'a']
#mengurutkan objek dalam list a tanpa mengubah urutan asli
a = ['a', 'i', 'u', 'e', 'o']
b = sorted(a)
c = sorted(a,reverse=True)
print('a =', a) #output: a = ['a', 'i', 'u', 'e', 'o']
print('b =', b) #output: b = ['a', 'e', 'i', 'o', 'u']
print('c =', c) #output: b = ['u', 'o', 'i', 'e', 'a']
#mengurutkan list berdasarkan panjang string
a = ['Asia', 'Antartika', 'Eropa', 'Afrika']
print('a =', a) #output: a = ['Asia', 'Antartika', 'Eropa', 'Afrika']
a.sort(key=len)
print('a =', a) #output: a = ['Asia', 'Eropa', 'Afrika', 'Antartika']
a.sort(key=len, reverse=True)
print('a =', a) #output: a = ['Antartika', 'Afrika','Eropa', 'Asia']
```