Bab 6. Functions

Selama ini kita sudah sering menggunakan perintah-perintah seperti `int()`, `str()`, `print()` dan sebagainya di dalam Python. Perintah-perintah tersebut secara spesifik disebut dengan function atau fungsi. Di dalam Python, sebuah function bisa jadi sudah tersedia dan langsung bisa digunakan, seperti misalnya `int()`, `str()`, `print()` dll, tanpa harus kita mendefinisikan atau membuatnya terlebih dahulu. Function jenis ini dinamakan *built in functions*. Namun, ada juga function yang harus kita buat atau definisikan secara manual terlebih dahulu sebelum digunakan.

Jika kita perhatikan function `int()`, `str()` dan sebagainya, maka ciri khas sebuah function adalah adanya tanda kurung () setelah nama functionnya, misalnya `print()`, `input()`, `str()` dll.

Gambaran sebuah function di dalam sebuah program adalah seperti subprogram di dalam sebuah program yang bertugas untuk menjalankan serangkaian perintah secara spesifik.

Analogi sebuah function dalam program adalah seperti sebuah organisasi. Di dalam organisasi tersebut terdapat beberapa divisi yang masing-masing divisi memiliki tugas tertentu secara spesifik. Setiap kali pimpinan organisasi akan menyelesaikan sebuah pekerjaan, maka cukup pimpinan organisasi meminta divisi tertentu yang sesuai dengan jenis pekerjaannya untuk menyelesaikannya. Dengan adanya keberadaan divisi dalam sebuah organisasi tentunya akan sangat membantu si pimpinan tanpa harus menyelesaikannya sendiri. Demikian pula di dalam sebuah program komputer. Seorang programmer bisa membuat sebuah subprogram (function) yang berfungsi sebagaimana divisi dalam organisasi tadi. Sehingga manfaat adanya function adalah bisa lebih efisien dalam proses koding dan terstruktur, serta terdapat konsistensi dalam alur program. Setiap kali programmer membutuhkan sebuah function yang sudah dibuatnya untuk melakukan suatu proses, maka programmer cukup memanggil nama functionnya saja, tanpa harus koding lagi secara berulang-ulang dengan baris program yang sama.

Cara Mendefinisikan Function

Sebuah function dibuat dengan tujuan melakukan tugas atau menjalankan serangkaian perintah tertentu, sesuai yang ditentukan.

Perhatikan contoh program berikut ini, di mana sebuah function kita buat atau definisikan.

```
def hello():
    print('Hallo')
    print('Hallo Bandung')
    print('Ibukota Priangan')

hello()
hello()
```

Untuk mendefinisikan sebuah function, digunakan perintah `def` kemudian diikuti dengan nama function yang akan dibuat, selanjutnya tambahkan tanda kurungnya. Dalam contoh tersebut, nama function yang dibuat adalah `hello()`.

Blok perintah

```
print('Hallo')
print('Hallo Bandung')
```

```
print('Ibukota Priangan')
```

adalah serangkaian perintah yang dijalankan ketika function `hello()` tersebut dipanggil. Oleh karena itu ketika diberikan perintah

```
hello()
hello()
```

akan diperoleh output sebagai berikut:

```
Hallo
Hallo Bandung
Ibukota Priangan
Hallo
Hallo Bandung
Ibukota Priangan
```

dalam contoh tersebut, function `hello()` dipanggil sebanyak dua kali.

Contoh function `hello()` yang diberikan di atas merupakan function yang tidak memiliki parameter atau *argument*. Parameter dalam sebuah function berfungsi seperti halnya sebuah input bagi function tersebut supaya bisa bekerja atau berjalan dengan normal. Penulisan parameter sebuah function terletak di dalam tanda kurung setelah nama functionnya.

Perhatikan contoh function ke dua berikut ini:

```
def hello(nama):
    print('Hallo, selamat siang ' + nama)
    print('Nama Anda indah sekali')

hello('Rosihan Ari')
hello('Dwi Amalia')
hello('Faza Fauzan')
```

Pada contoh ke dua ini, function `hello()` memiliki sebuah parameter yaitu *nama*. Nilai dari parameter nama nantinya akan dicetak ketika function `hello()` ini dipanggil. Sehingga output dari program tersebut adalah

```
Hallo, selamat siang Rosihan Ari
Nama Anda indah sekali
Hallo, selamat siang Dwi Amalia
Nama Anda indah sekali
Hallo, selamat siang Faza Fauzan
Nama Anda indah sekali
```

Banyaknya parameter sebuah function bisa lebih dari satu sesuai kebutuhan yang diinginkan. Sebagai contoh, perhatikan contoh ke tiga berikut ini:

```
def penjumlahan(bil1, bil2):
    hasil = bil1 + bil2
    print(bil1,'+', bil2,'=', hasil)

penjumlahan(3, 5)
penjumlahan(2, -4)
```

Function `penjumlahan()` pada contoh di atas digunakan untuk menjumlahkan dua buah bilangan. Dalam hal ini function tersebut memiliki dua buah parameter yaitu *bil1* dan *bil2* sebagai kedua bilangan yang akan dijumlahkan (Catatan: antar parameter dipisahkan dengan tanda koma). Setelah dijumlahkan, selanjutnya hasil penjumlahannya akan ditampilkan. Adapun output dari program setelah dijalankan adalah

3 + 5 = 8

2 + -4 = -2

Parameter Function dengan Nilai Default

Sebuah parameter function dapat diberikan nilai default tertentu. Akibat dari pemberian nilai default ini adalah apabila parameter tersebut tidak diberikan nilai ketika pemanggilan function, maka parameter tersebut akan diberikan nilai sesuai nilai defaultnya. Perhatikan contoh berikut ini:

```
def jumlah(bil1, bil2=0):
    hasil = bil1 + bil2
    print(bil1, '+', bil2, '=', hasil)
```

Pada contoh tersebut, parameter `bil2` diberikan nilai default 0. Artinya, jika nilai parameter `bil2` ini tidak diberikan ketika pemanggilan function `jumlah()` maka `bil2` diberi nilai 0. Sehingga apabila diberikan perintah berikut ini

```
jumlah(3, 6)
jumlah(10)
```

akan dihasilkan output

```
3 + 6 = 9
10 + 0 = 10
```

Penting untuk diperhatikan, bahwa parameter yang diberikan nilai default harus diletakkan setelah parameter tanpa nilai default secara urutan penulisannya. Jika tidak, maka akan dihasilkan pesan error seperti contoh berikut ini:

```
def jumlah(bil1, bil2=0, bil3):
    hasil = bil1 + bil2 + bil3
    print(hasil)
```

```
  File "<ipython-input-9-6f864c6339ad>", line 1
    def jumlah(bil1, bil2=0, bil3):
               ^
SyntaxError: non-default argument follows default argument
```

Dalam contoh tersebut, parameter misalkan `bil2` diberikan nilai default 0. Akan tetapi karena letaknya di depan parameter `bil3` yang tidak bernilai default, maka akan muncul error. Oleh karena itu, solusinya adalah dengan mengubah urutan penulisan parameternya menjadi seperti ini:

```
def jumlah(bil1, bil3, bil2=0):
    hasil = bil1 + bil2 + bil3
    print(hasil)
```

Function dengan *Dynamic Parameter*

Perhatikan function `print()`. Function `print()` ini dapat dipanggil dengan bermacam-macam jumlah parameter, misalnya:

```
print()
print('hello')
print('hello', 'apa', 'kabar')
print('hello', 'apa', 'kabar', 'semuanya')
print('hello', 'apa', 'kabar', 'semuanya', '???')
```

```
hello
hello apa kabar
hello apa kabar semuanya
hello apa kabar semuanya ???
```

Berdasarkan contoh di atas, tampak bahwa banyaknya parameter `print()` bisa bermacam-macam. Sifat parameter seperti ini dinamakan *dynamic parameter*. Bagaimana cara membuat function dengan sifat parameter seperti ini? Untuk mendefinisikan sebuah parameter yang sifatnya dynamic, yaitu cukup dengan menambahkan tanda * di depan nama parameternya. Berikut ini contohnya:

```
def rerata(*myData):

    # init values
    sum = 0
    i = 0

    # menjumlahkan semua data dalam myData
    for data in myData:
        sum += data
        i += 1

    # hitung rata-rata
    rata2 = sum/i
    print('Rata-ratanya: ', rata2)
```

Function `rerata()` digunakan untuk mencari rata-rata beberapa bilangan. Dalam hal ini parameter yang digunakan cukup satu saja, yaitu `myData`. Namun, data dalam `myData` ini bisa lebih dari satu buah maka perlu ditambahkan tanda *. Tanda * ini digunakan menandai bahwa parameter tersebut bertipe data *tuple*. Pembahasan lebih lanjut tentang salah satu tipe data dinamis ini ada di bab yang lainnya.

Selanjutnya sebelum menghitung nilai rata-ratanya, diperlukan menjumlahkan dulu semua data dalam `myData`. Dalam hal ini dapat menggunakan perulangan for. Setelah didapatkan hasil penjumlahan semua datanya, barulah dapat dicari rata-ratanya.

Contoh pemanggilan function `rerata()` adalah sebagai berikut:

```
rerata(1, 2, 3, 4, 5)
```

```
Rata-ratanya:  3.0
```

Function dengan *Return Value (Non-void Function)*

Sebuah function dapat pula dibuat supaya menghasilkan suatu nilai (value) bertipe data tertentu sebagai output dari function tersebut. Selanjutnya output atau *value* yang dihasilkan function tersebut dapat digunakan untuk proses yang lainnya. Function jenis ini disebut dengan *non-void function*. Sedangkan kebalikannya, function yang tidak mengembalikan sebuah nilai dinamakan *void function*. Adapun contoh *void function* sudah diberikan pada contoh-contoh sebelumnya.

Ciri khas *non-void function* adalah terdapatnya perintah `return` di dalam function. Perhatikan contoh program berikut ini.

```
def penjumlahan(bil1, bil2):
    hasil = bil1 + bil2
    return hasil

a = 4
b = 3
# menampilkan hasil penjumlahan 4 + 3
print(a, '+', b, '=', penjumlahan(a, b))

a = penjumlahan(2, 4)
b = penjumlahan(-1, 3)
# menampilkan hasil penjumlahan (2+4)+(-1+3)
print(a, '+', b, '=', penjumlahan(a, b))

a = penjumlahan(penjumlahan(1, 2), 4)
b = penjumlahan(penjumlahan(-1, 4), penjumlahan(4, 5))
# menampilkan hasil penjumlahan ((1+2)+4)+((-1+4)+(4+5))
print(a, '+', b, '=', penjumlahan(a, b))
```

Function `penjumlahan()` pada program di atas sedikit dimodifikasi dari contoh sebelumnya. Jika diperhatikan, maka di akhir bagian function diberikan perintah `return hasil`. Maksud dari perintah ini adalah menjadikan nilai dari variabel hasil sebagai output dari function `penjumlahan()`.

Sekarang kita lihat baris pada program berikut ini

```
a = 4
b = 3
print(a, '+', b, '=', penjumlahan(a, b))
```

Baris program tersebut bermaksud untuk menjumlahkan dua bilangan `a` dan `b`, di mana masing-masing diberi nilai 4 dan 3. Di dalam perintah print() terdapat pemanggilan terhadap function `penjumlahan(a, b)` sehingga dalam hal ini function tersebut akan menjumlahkan bilangan 4 dan 3. Dari proses penjumlahan didapatkan `hasil = 7`. Nilai dari variabel hasil inilah yang menjadi output dan akan dikembalikan kepada perintah `penjumlahan(a, b)` tersebut. Sehingga perintah `penjumlahan(a, b)` bernilai 7. Akibatnya, akan dihasilkan tampilan

4 + 3 = 7.

Selanjutnya pada baris:

```
a = penjumlahan(2, 4)
b = penjumlahan(-1, 3)
print(a, '+', b, '=', penjumlahan(a, b))
```

Variabel `a` diberi nilai dari hasil penjumlahan 2 dan 4 yaitu 6, dan b diberi nilai hasil dari penjumlahan -1 dan 3 yaitu 2. Selanjutnya `print()` akan menampilkan hasil penjumlahan `a` dan `b` sebagai berikut

6 + 2 = 8.

Sedangkan pada baris program:

```
a = penjumlahan(penjumlahan(1, 2), 4)
b = penjumlahan(penjumlahan(-1, 4), penjumlahan(4, 5))
print(a, '+', b, '=', penjumlahan(a, b))
```

Nilai `a` diberi nilai dari hasil penjumlahan 3 dan 4, di mana 3 diperoleh dari hasil penjumlahan 1 dan 2. Sehingga dalam hal ini, nilai `a` bernilai 7. Sedangkan `b`, diberi nilai dari hasil penjumlahan 3 dan 9, di mana 3 diperoleh dari hasil penjumlahan -1 dan 4, dan 9 diperoleh dari hasil penjumlahan 4 dan 5. Oleh karena itu, nilai `b` adalah 12. Akibatnya output yang muncul adalah

7 + 12 = 19.

Lebih Lanjut dengan Function *print()*

Khusus untuk function `print()` yang sering kita gunakan, ada parameter tambahan bersifat optional yang bisa kita tambahkan. Selama ini, jika kita menggunakan `print()` maka outputnya akan menghasilkan baris baru di bawahnya. Sebagai contoh:

```
print('Hello')
print('World')
```

akan menghasilkan dua baris string ketika dijalankan

```
Hello
World
```

Hal tersebut disebabkan karena ketika perintah `print()` diberikan, secara default akhir dari string terdapat karakter `\n` (*newline*) secara implisit. Karakter *newline* ini dapat kita ubah dengan karakter apapun dengan cara menambahkan parameter `end='...'` di dalam `print()`, di mana titik-titiknya bisa diisi dengan sembarang karakter. Contoh:

```
print('Hello', end=' ')
print('World')
```

Di dalam contoh tersebut, akhir string 'Hello' ditambahkan karakter spasi kosong, sehingga dalam hal ini string 'World' yang dicetak setelahnya akan terletak di sebelah kanan 'Hello' didahului dengan karakter spasi kosong sebelumnya.

```
Hello World
```

Mengimport Function dari File Lain

Apabila sebuah function sudah dibuat/didefinisikan dan disimpan ke dalam sebuah file, maka kita bisa memanggil function tersebut dari file yang lainnya. Sebagai contoh, misalkan diberikan sebuah file program Python dengan nama '**hello.py**' (dibuat melalui Spyder atau IDE lainnya) yang di dalamnya terdapat sebuah function `hello1()` dan `hello2()` berikut ini

```
def hello1():
    print('Hello world')

def hello2():
    print('Hello everybody');
```

Untuk mengimport semua function yang ada dalam file hello.py tersebut, caranya adalah dengan memberikan perintah berikut ini:

```
from hello import *
```

Perlu diingat bahwa, letak file yang diimport harus dalam satu folder yang sama dengan file yang mengimportnya.

Tanda * dalam perintah import tersebut bermakna bahwa yang akan dipanggil adalah semua function yang ada di dalam file tersebut. Adapun contoh penggunaan import secara lengkap, sekaligus dengan pemanggilan function-functionnya adalah sebagai berikut:

```
from hello import *
hello1()
hello2()
```

Adapun alternatif cara lain selain menggunakan perintah `from ... import ...` untuk mengimport module dari script Python adalah dengan perintah import yaitu:

```
import nama_module
```

Contoh untuk mengimport module dalam file hello.py menggunakan cara ini adalah

```
import hello
hello.hello1()
hello.hello2()
```

Makna dari perintah `hello.hello1()` adalah mengakses function `hello1()` yang ada dalam module hello.py. Demikian juga untuk perintah `hello.hello2()`.

Selanjutnya bagaimana jika file hello.py tersebut terletak dalam subfolder misalnya 'myfunctions'? Jika demikian halnya, maka perlu menuliskan nama subfolder di depan nama filenya. Selanjutnya diberikan tanda . (titik), misalnya:

```
from myfunctions.hello import *
```

Sedangkan apabila dengan perintah import adalah

```
import modules.hello
```

Sehingga untuk perintah import tersebut, apabila hendak menggunakan function `hello1()`, maka perintah yang digunakan adalah:

```
modules.hello.hello1()
```

Agak sedikit panjang perintahnya bukan? Namun, meskipun demikian dapat dibuat alias yang lebih sederhana sehingga menyederhanakan perintah. Caranya adalah

```
import modules.hello as h
```

```
h.hello1()
```

```
Hello world
```

Variabel Lokal dan Global (*Scope*)

Pada sebuah program yang di dalamnya terdapat function, kita harus memahami karakteristik penggunaan variabelnya. Dalam hal ini, kita harus memahami apa itu variabel lokal dan global. Untuk bisa memahaminya, perhatikan tiga contoh program berikut ini.

Program Pertama

```
def myFunction1():
    a = 4

myFunction1()
print(a)
```

Program Ke Dua

```
def myFunction2():
    print(a)

a = 4
myFunction2()
```

Program Ke Tiga

```
def myFunction1():
    a = 4

def myFunction2():
    print(a)

myFunction1()
myFunction2()
```

Pada program pertama, terdapat function `myFunction1()` yang di dalamnya ada pendefinisian variabel `a` yang kemudian diberi nilai 4. Selanjutnya di luar function, terdapat perintah untuk mencetak nilai variabel `a`. Apabila program pertama dijalankan, maka akan terdapat error

```
NameError: name 'a' is not defined
```

Kesimpulan yang bisa diambil dari program pertama adalah ternyata variabel `a` yang didefinisikan di dalam function `myFunction1()` tidak dikenal di luar function tersebut.

Selanjutnya di dalam program ke dua, di luar function `myFunction2()` didefinisikan variabel `a` yang diberi nilai 4. Selanjutnya function `myFunction2()` dipanggil di mana akan mencetak nilai variabel `a` yaitu 4. Dalam hal ini, variabel `a` meskipun didefinisikan di luar function `myFunction2()` akan tetapi tetap dikenali di dalam function tersebut yang diindikasikan melalui tidak munculnya error sebagaimana pada program pertama.

Sedangkan pada program ke tiga, ketika dijalankan maka juga akan menghasilkan error

```
NameError: name 'a' is not defined
```

Adapun penyebab error tersebut adalah bahwa variabel `a` yang didefinisikan di dalam function `myFunction1()` tidak dikenal di dalam `myFunction2()`.

Berdasarkan ketiga contoh program yang telah diberikan, dapat disimpulkan bahwa variabel yang didefinisikan di dalam sebuah function adalah bersifat lokal yaitu hanya bisa dikenal dan digunakan di dalam function itu sendiri. Hal ini dapat dilihat pada program pertama dan ke tiga. Sedangkan variabel yang didefinisikan di program utama (bukan di dalam function) maka juga akan dikenali di dalam function apapun. Hal ini dapat dilihat pada program ke dua. Sehingga dalam hal ini sifat variabel tersebut adalah global atau dikenali di semua bagian dalam program.

Selanjutnya, akan diberikan sebuah contoh program berikut ini:

```
def myFunction1():
    a = 4

a = 10
myFunction1()
print(a)
```

Jika program tersebut dijalankan, maka akan dihasilkan output 10. Mengapa demikian? Mengapa tidak 4 yang dicetak?

Pada program tersebut, mula-mula variabel `a` diberi nilai 10 di dalam program utama. Selanjutnya terjadi pemanggilan function `myFunction1()` di mana pada function ini ada proses *assignment* terhadap variabel `a` dengan nilai 4. Setelah itu, nilai variabel `a` dicetak dan akan memberikan hasil 10. Apabila program tersebut diperhatikan, maka ada dua pendefinisian variabel `a`, yaitu di program utama dan di dalam function `myFunction1()`. Meskipun nama variabelnya sama, akan tetapi keduanya berbeda yang disebabkan sifat dari keduanya. Variabel `a` yang didefinisikan di dalam program utama adalah bersifat global. Sebaliknya, variabel `a` yang didefinisikan di dalam function `myFunction1()` bersifat lokal. Oleh karena itu, ketika perintah `print(a)` di dalam program utama dijalankan maka variabel `a` di sini adalah variabel `a` yang bersifat global.

Pertanyaan berikutnya adalah bagaimana caranya supaya nilai `a` yang awalnya diberi nilai 10 pada program di atas, yang dalam hal ini sebagai variabel global bisa diubah nilainya menjadi 4 di dalam function `myFunction1()`? Untuk melakukan hal ini caranya adalah dengan memberikan label global di pada variabel `a` di dalam `myFunction1()` yang bertujuan untuk memberi tanda bahwa variabel `a` yang disebutkan di dalam function tersebut adalah variabel a yang bersifat global, sehingga menjadi

```
def myFunction1():
    global a
    a = 4

a = 10
myFunction1()
print(a)
```

Setelah perubahan pada program akan dihasilkan output 4, yang dalam hal ini merupakan nilai variabel global `a` yang sudah diubah nilainya.