FUNGSI PADA PYTHON

1. Pendahuluan

Fungsi (Function) adalah suatu program terpisah dalam blok sendiri yang berfungsi sebagai sub-program (modul program) yang merupakan sebuah program kecil untuk memproses sebagian dari pekerjaan program utama.

Keuntungan menggunakan fungsi :

1. Program besar dapat di pisah-pisah menjadi program-program kecil melalui function.
2. Kemudahan dalam mencari kesalahan-kesalahan karena alur logika jelas dan kesalahan dapat dilokalisasi dalam suatu modul tertentu.
3. Memperbaiki atau memodifikasi program dapat dilakukan pada suatu modul tertentu saja tanpa menggangu keseluruhan program.
4. Dapat digunakan kembali (Reusability) oleh program atau fungsi lain.
5. Meminimalkan penulisan perintah yang sama.

Kategori Fungsi

1. **Standard Library Function**

adalah fungsi-fungsi yang telah disediakan oleh Interpreter Python dalam file-file atau librarynya.
Misalnya: raw_input(), input(), print(), open(), len(), max(), min(), abs() dll.

2. **Programme-Defined Function**

Adalah **function** yang dibuat oleh programmer sendiri. **Function** ini memiliki nama tertentu yang unik dalam program, letaknya terpisah dari program utama, dan bisa dijadikan satu ke dalam suatu library buatan programmer itu sendiri.

Dalam python terdapat dua perintah yang dapat digunakan untuk membuat sebuah fungsi, yaitu *def* dan *lambda*. *def* adalah perintah standar dalam python untuk mendefinisikan sebuah fungsi. Tidak seperti function dalam bahasa pemrograman compiler seperti C/C++, *def* dalam python merupakan perintah yang executable, artinya function tidak akan aktif sampai python me-running perintah *def* tersebut. Sedangkan *lambda*, dalam python lebih dikenal dengan nama *Anonymous Function* (Fungsi yang tidak disebutkan namanya). *Lambda* bukanlah sebuah perintah (statemen) namun lebih kepada ekspresi (expression).

2. Mendeklarasikan dan Memakai Fungsi

Statemen *def*

Statemen *def* digunakan untuk mendeklarasikan fungsi. Sedangkan statemen *return* digunakan untuk mengembalikan suatu nilai kepada bagian program yang memanggil fungsi. Bentuk umum untuk mendeklarasikan fungsi adalah sebagai berikut :

```
def <nama_fungsi>(arg1, arg2, arg3, …,argN) :
     <statemen-statemen>
```

Sebuah fungsi diawali dengan statemen *def* kemudian diikuti oleh sebuah *nama_fungsi* nya. Sebuah fungsi dapat memiliki daftar argumen (parameter) ataupun tidak. Tanda titik dua (:) menandakan awal pendefinisian tubuh dari fungsi yang terdiri dari statemen-statemen.

Tubuh fungsi yang memiliki statemen return :

```
def <nama_fungsi>(arg1, arg2, arg3, …,argN) :
     <statemen-statemen>
     …..
     return <value>
```

Statemen return dapat diletakkan di bagian mana saja dalam tubuh fungsi. Statemen *return* menandakan akhir dar pemanggilan fungsi dan akan mengirimkan suatu nilai balik kepada program yang memanggil fungsi tersebut. Statemen *return* bersifat opsional, artinya jika sebuah fungsi tidak memiliki statemen *return*, maka sebuah fungsi tidak akan mengembalikan suatu nilai apapun.

Contoh penggunaan fungsi :

```
>>> def ucapan():
...     print "Anda sedang menggunakan fungsi"
...
>>> ucapan()
Anda sedang menggunakan fungsi
```

Pernyataan *def* mendefinisikan sebuah fungsi dengan nama *ucapan. Fungsi ucapan* tidak memiliki daftar argumen dan tidak meminta nilai kembalian. Pendefinisian fungsi ucapan diakhiri dengan tanda (:), kemudian diikuti oleh statemen *print* yang menjadi isi dari tubuh fungsi. Lalu untuk memanggil fungsi ucapan() kita gunakan perintah,

```
     <nama_fungsi>()
contohnya,
     ucapan()
```

Contoh program dengan melibatkan nilai balik (return):

```
def perkalian(a,b):
     c = a*b
     return c

# Program Utama
print( perkalian(5,10))

output:
     50
```

Pada contoh diatas, sebuah fungsi dengan nama perkalian(), memiliki dua buah argumen yaitu a dan b. Isi dari fungsi tersebut adalah melakukan perhitungan perkalian yang diambil dari nilai a dan b, yang di simpan ke dalam variabel c. Nilai dari c lah yang akan dikembalikan oleh fungsi dari hasil pemanggilan fungsi melalui statemen perkalian(5, 10). Dimana nilai 5 akan di simpan dalam variabel a dan nilai 10 akan disimpan dalam variabel b.

Scope Variabel

Scope variabel atau cakupan variabel merupakan suatu keadaan dimana pendeklarasian sebuah variabel di tentukan. Dalam scope variabel dikenal dua istilah yaitu *local* dan *global* . Variabel disebut *local* ketika variabel tersebut didefinisikan didalam sebuah fungsi (*def*). Artinya, variabel tersebut hanya dapat di gunakan dalam cakupan fungsi tersebut saja. Dan jika sebuah variabel didefinisikan diluar fungsi maka variabel tersebut bersifat *global*. Artinya, variabel tersebut dapat digunakan oleh fungsi lain atau pun program utamanya.

Contoh penggunaan scope variabel :

```
def contohScope(X):
     X = 10
     print "Nilai X di dalam fungsi, x = ", X

# program utama
X = 30
print "Nilai x di luar fungsi, x = ", X
contohScope(X)
```

Output :

```
Nilai x di luar fungsi, x =  30
Nilai X di dalam fungsi, x =  10
```

Pada contoh diatas, variabel X didefinisikan di dua tempat yaitu di dalam fungsi contohScope() dan di dalam program utama. Ketika nilai X awal di beri nilai 30, kemudian di cetak, nilai X masih bernilai 30. Namun ketika kita memanggil fungsi contohScope() dengan mengirim parameter X yang bernilai 30, terlihat bahwa nilai X yang berlaku adalah nilai X yang didefinisikan didalam fungsi tersebut. Atau nilai X yang bernilai 10. ini terbukti bahwa variabel X yang di cetak dalam fungsi contohScope() merupakan variabel local yang didefinisikan didalam fungsi, bukan variabel X global yang dicetak di luar fungsi.

Contoh lain

```
# fungsi mulai di sini
def swap(x, y):
     print "Dalam fungsi:"
     print "\tSebelum proses:"
     print "\t\tNilai x", x
     print "\t\tNilai y", y
     z = x
     x = y
     y = z
     print "\tSetelah proses:"
     print "\t\tNilai x", x
     print "\t\tNilai y", y

# program utama mulai di sini
x = 12
y = 3
print "Sebelum memanggil fungsi, x bernilai", x
print "Sebelum memanggil fungsi, y bernilai", y
swap(x,y)
print "Setelah memanggil fungsi, x bernilai", x
print "Setelah memanggil fungsi, y bernilai", y
```

Output :

```
Sebelum memanggil fungsi, x bernilai 12
Sebelum memanggil fungsi, y bernilai 3
Dalam fungsi:
     Sebelum proses:
          Nilai x 12
          Nilai y 3
```

```
        Setelah proses:
             Nilai x 3
             Nilai y 12
    Setelah memanggil fungsi, x bernilai 12
    Setelah memanggil fungsi, y bernilai 3
```

Dalam lingkungan program utama variabel x dan y diisi dengan nilai 12 dan 3, secara berurutan. Ketika program utama memanggil fungsi swap nilai variabel ini disalin kedalam variabel x dan y dalam lingkungan fungsi, yang kebetulan sama namanya dengan nama variabel dalam progam utama. Python akan membuat alokasi memori tersendiri untuk fungsi. Karena alokasi memorinya berbeda, maka perubahan yang terjadi pada variabel dalam fungsi, katakanlah x, tidak akan mengubah variabel x yang terdapat dalam program utama. Jadi bisa dikatakan variabel x dalam fungsi swap hanya mempunyai scope dalam fungsi itu sendiri. Variabel program utama tetap nilainya, meskipun variabel dalam fungsi swap berubah nilainya.

Statemen *Lambda*

Selain statemen *def*, Python juga menyediakan suatu bentuk ekspresi yang menghasilkan objek fungsi. Karena kesamaannya dengan tools dalam bahasa Lisp, ini disebut lambda .Seperti *def*, ekspresi ini menciptakan sebuah fungsi yang akan dipanggil nanti, tapi mengembalikan fungsi dan bukan untuk menetapkan nama. Inilah sebabnya mengapa kadang-kadang lambda dikenal sebagai anonim (yakni, tidak disebutkan namanya) fungsi. Dalam prakteknya, mereka sering digunakan sebagai cara untuk inline definisi fungsi, atau untuk menunda pelaksanaan sepotong kode.

Bentuk umum *lambda* adalah kata kunci *lambda*, diikuti oleh satu atau lebih argumen (persis seperti daftar argumen dalam tanda kurung di *def* header), diikuti oleh ekspresi setelah tanda titik dua:

```
lambda argument1, argument2,... argumentN :expression using arguments
```

lambda memiliki perbedaan dengan *def* antara lain :

1. ***lambda* adalah sebuah ekspresi, bukan pernyataan.** Karena ini, sebuah *lambda* dapat muncul di tempat-tempat *def* tidak diperbolehkan oleh sintaks Python-di dalam daftar harfiah atau pemanggilan fungsi argumen, misalnya. Sebagai ekspresi, lambda mengembalikan nilai (fungsi baru) yang opsional dapat diberi nama. Sebaliknya, pernyataan def selalu memberikan fungsi baru ke nama di header, bukannya kembali sebagai hasilnya.
2. **tubuh lambda adalah ekspresi tunggal, bukan satu blok statemen**. Tubuh *lambda* sama dengan apa yang akan dimasukkan ke dalam statemen *return* dalam tubuh *def.*

Contoh penggunaan lambda :

```
>>> f = lambda x, y, z: x + y + z
>>> f(10,20,30)
60
```

Contoh 2 :

```
>>> def nama():
...     gelar = 'Sir'
...     aksi = (lambda x: gelar + ' ' + x)
```

```
          ...     return aksi
          ...
          >>> act = nama()
          >>> act('Robin')
          'Sir Robin'

     contoh 3 :

     >>> z = (lambda a = "tic", b = "tac", c = "toe" : a + b + c)
     >>> z("ZOO")
     'ZOOtactoe'
```

3. Fungsi Rekursif

Fungsi Rekursif merupakan suatu fungsi yang memanggil dirinya sendiri. Artinya, fungsi tersebut dipanggil di dalam tubuh fungsi itu sendiri. Tujuan di lakukan rekursif adalah untuk menyederhanakan penulisan program dan menggantikan bentuk iterasi. Dengan rekursi, program akan lebih mudah dilihat.

Mencari nilai faktorial dari suatu bilangan bulat positif adalah salah satu pokok bahasan yang memudahkan pemahaman mengenai fungsi rekursif. Berikut adalah fungsi faktorial yang diselesaikan dengan cara biasa :

Konsep faktorial,

```
N ! = faktorial(N) = 1 * 2 * 3 … * N
```

Dalam pemrograman konsep dari faktorial seperti berikut,

```
faktorial(N)   = N!
               = N * (N-1)!
               = N * (N-1) * (N-2)!

               = N * (N-1) * (N-2) … * 3 * 2 * 1
```

Program mencari nilai faktorial :

```
# Fungsi Rekursif faktorial
def faktorial(nilai):
     if nilai <= 1:
          return 1
     else:
          return nilai * faktorial(nilai - 1)

#Program utama
for i in range(11):
     print "%2d ! = %d" % (i, faktorial(i))
```

```
Output :

     0 ! = 1
     1 ! = 1
     2 ! = 2
     3 ! = 6
     4 ! = 24
     5 ! = 120
     6 ! = 720
     7 ! = 5040
     8 ! = 40320
     9 ! = 362880
     10 ! = 3628800
```

Selain Faktorial, pencarian deret fibonachi juga merupakan salah satu contoh yang mengimplementasikan fungsi rekursif, berikut adalah program mencari nilai deret fibonachi

Konsep Fibonaci :

fibonaci merupakan deret yang dimulai dari angka 0 dan 1. dan untuk nilai deret yang lain, dihasilkan dari penjumlahan dua bilangan sebelumnya,

0, 1, 1, 2, 3, 5, 8, 13, 21,.....

```
     fibonacci( 0 ) = 0
     fibonacci( 1 ) = 1
     fibonacci( n ) = fibonacci( n – 1 ) + fibonacci( n – 2 )
```

```
program Fibonaci :

          # Fungsi Fibonacci
          def fibonacci(n):
          if n < 0:
               print "Tidak ada bilangan yang bernilai negatif"

          if n == 0 or n == 1:
               return n
          else:
               return fibonacci(n-1) + fibonacci(n-2)


          # Program utama
          nilai = int(raw_input("Masukkan sebuah bilangan : " ))
          hasil = fibonacci(nilai)
          print "Fibonacci(%d) = %d" % (nilai,hasil)
Output :

          Masukkan sebuah bilangan : 20
          Fibonacci(20) = 6765
```

Melewatkan Argumen dengan Kata Kunci

Kalau kita perhatikan kembali fungsi perkalian sebelumnya, proses penyalinan ke variabel lokal sesuai dengan urutan deklarasi fungsi yang kita panggil. Jika fungsi perkalian kita panggil dengan memberi pernyataan perkalian(10,8), maka nilai 10 akan disalin ke variabel x dan nilai 8 ke variabel y. Kadang-kadang ini agak menyulitkan jika kita membuat fungsi dengan jumlah variabel yang cukup banyak, sementara urutannya harus tepat. Solusinya adalah dengan menyebutkan kata-kunci (keyword) yang kita pakai pada saat mendefinisikan fungsi.

Kita ubah sedikit program perkalian kita agar pembahasan di bagian ini lebih jelas. Perhatikan program di bawah.

```
def perkalian(x, y):
   "Mengalikan dua bilangan"
   z = x * y
   print "Nilai x =",x
   print "Nilai y =",y
   print "x * y =",z

# program utama mulai di sini
perkalian(10,2)
print
perkalian(y=15,x=5)
```

Hasilnya:

```
Nilai x = 10
Nilai y = 2
x * y = 20

Nilai x = 5
Nilai y = 15
x * y = 75
```

Dengan menyebutkan kata kunci yang kita buat saat mendeklarasikan program kita dapat mengubah urutan penyalinan argumen. Akan tetapi Anda harus berhati-hati ketika menyebutkan kata-kunci, karena tidak boleh ada duplikasi. Panggil fungsi perkalian dengan pernyataan perkalian(15,x=5), maka Anda akan mendapatkan pesan kesalahan sbb.:

```
Traceback (innermost last):
  File "./listing8.py", line 13, in ?
    perkalian(15,x=5)
TypeError: keyword parameter redefined
```

Hasil ini menunjukkan pada kita bahwa nama x sudah dipakai. Dengan melihat pada definisi fungsi yang telah dibuat, parameter pertama adalah x dan kedua adalah y. Jadi ketika kita panggil dengan menyebutkan parameter kedua sebagai x juga akan terjadi kesalahan.

Nilai Awal Argumen

Dalam proses interaksi dengan pengguna program kadangkala program memberikan pilihan tertentu, yang sering disebut dengan nilai bawaan (default). Nilai awal argumen ini bisa kita berikan saat kita membuat definisi fungsi. Lihat cara mendeklarasikan nilai awal argumen ini:

```
def login(username="admin", password="aa"):
   print "Your username ",username
   print "Your password ",password
```

```
    print

login()
login("tamu")
login("tamu", "katakunci")
```

Sekarang proses pemanggilan fungsi tidak perlu menyebutkan argumennya secara lengkap, jika kita tidak perlu mengubah nilai default yang telah diberikan.

```
Your username  admin
Your password  aa

Your username  tamu
Your password  aa

Your username  tamu
Your password  katakunci
```

Dengan membandingkan antara isi program dan hasilnya di atas, dapat kita simpulkan bahwa penyalinan argumen tetap mengikuti kaidah urutan pada saat dideklarasikan.

Anda tidak diperbolehkan mendefinisikan fungsi seperti ini:

```
def login(username="admin", password):
    print "Your username ",username
    print "Your password ",password
    print
```

Akan tetapi Anda bisa mendeklarasikan fungsi seperti potongan program berikut.

```
def login(username, password="aa"):
    print "Your username ",username
    print "Your password ",password
    print
```

Jadi nilai default hanya boleh diberikan kepada deretan akhir parameter. Setelah pemberian nilai default, semua parameter di belakangnya juga harus diberi nilai default. Satu catatan, nilai awal argumen akan dievaluasi pada saat dideklarasikan. Perhatikan contoh berikut.

```
usernm="admin"
passwd="aa"
def login(username=usernm, password=passwd):
    print "Your username ",username
    print "Your password ",password
    print

usernm="tamu"
passwd="cc"

login()
```

Hasilnya:

```
Your username  admin
Your password  aa
```

Jumlah Argumen yang Berubah

Terdapat dua lambang khusus dalam Python untuk menerima argumen dengan jumlah yang berubah-ubah. Lambang pertama adalah *nama_argumen. Dengan memakai lambang ini pada

deklarasi fungsi, Python akan mengenali argumen selain argumen formal sebagai tuple. Lihat kode berikut ini:

```
def guest(name, password, *hobby):
   print "Your name    :",name
   print "Your password:",password
   print "Hobby Anda   :",hobby

guest("tamu", "katakunci", "memancing", "membaca", "olahraga")
```

Hasilnya:

```
Your name    : tamu
Your password: katakunci
Hobby Anda   : ('memancing', 'membaca', 'olahraga')
```

Untuk memanggil fungsi yang mempunyai deklarasi seperti ini, kita cukup memberikan daftar argumen seperti argumen biasa.

Lambang kedua adalah **nama_argumen. Dengan lambang ini argumen yang diterima oleh fungsi akan dikenali sebagai dictionary. Lihat contoh berikut:

```
def guest(name, password, **other):
   print "Your name    :",name
   print "Your password:",password
   print "Lain-lain    :",other

guest("tamu", "katakunci", sex="laki-laki", umur=18, hobby="membaca")
```

Hasilnya:

```
Your name    : tamu
Your password: katakunci
Lain-lain    : {'sex': 'laki-laki', 'hobby': 'membaca', 'umur': 18}
```

Untuk memanggil fungsi dengan deklarasi seperti ini, kita harus menyebutkan daftar argumen beserta kata-kuncinya.

Jika Anda ingin menggunakan dua lambang ini secara bersamaan Anda harus mendahulukan *nama_argumen daripada **nama_argumen.

```
def guest(name, password, *hobby, **other):
   print "Your name    :",name
   print "Your password:",password
   print "Hobby Anda   :",hobby
   print "Lain-lain    :",other

guest("tamu", "katakunci", "single", "membaca", sex="laki-laki", umur=18)
```

Hasil eksekusi program:

```
Your name    : tamu
Your password: katakunci
Hobby Anda   : ('single', 'membaca')
Lain-lain    : {'sex': 'laki-laki', 'umur': 18}
```

Contoh :

```
>>> def cetak1():
        print ‘Hello World’
```

```
>>> def cetak2(n):
        print n
>>> cetak1()
hallo world
>>> cetak2(123)
123
>>> cetak2('apa kabar?')
apa kabar

>>> def cetak3(x,y,z):
        print x,y,z
>>> def cetak4(x,y,z=4):
        print x,y,z
>>> cetak3(1,2,3)
1 2 3
>>> cetak4(1,2)
1 2 4
>>> cetak4(1,2,3)
1 2 3
```

- Fungsi pada Python
- Fungsi Aritmatik
- Fungsi Intrepeter
- Fungsi Rekursif

- Pemanggilan pada fungsi
- Membuat fungsi baru
- Alur eksekusi program
- Menggunakan parameter
- Variabel lokal dalam fungsi