

## Pertemuan3

# *Fungsi pada Python*

### **Objektif:**

1. Mahasiswa mengetahui dan memahami fungsi dalam Python
2. Mahasiswa mengetahui bentuk umum dari fungsi
3. Mahasiswa dapat menggunakan dan mendeklarasikan fungsi
4. Mahasiswa dapat membuat program sederhana untuk fungsi

## P3.1 Teori

### Pendahuluan

Fungsi digunakan untuk mengumpulkan beberapa perintah yang sering dipakai dalam sebuah program. 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. Dengan memakai fungsi, program yang kita buat menjadi lebih terstruktur. Lebih mudah diikuti oleh orang lain yang membaca program kita. Dan yang paling penting adalah mempersingkat waktu yang kita perlukan untuk mengembangkan suatu perangkat lunak. Karena perangkat lunak yang kita buat, bisa jadi memakai komponen-komponen yang sama.

Seperti layaknya sebuah bahasa pemrograman, Python juga memberikan fasilitas pembuatan fungsi yang sangat bagus. Konsep fungsi dalam Python sama dengan bahasa pemrograman C/C++. Python menganggap fungsi dan prosedur adalah sesuatu yang sama, dalam artian cara mendeklarasikan fungsi dan prosedur adalah sama. Hanya bedanya, kalau fungsi mengembalikan suatu nilai setelah proses sedangkan prosedur tidak.

#### Keuntungan menggunakan fungsi :

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

#### Kategori Fungsi

##### ➤ 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.

## ➤ Programme-Defined Function

Adalah fungsi yang dibuat oleh programmer sendiri. Fungsi 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).

## 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 dari 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`.

### 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 terkadang *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'
```

## 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. 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

```

## 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,

$$N ! = \text{faktorial}(N) = 1 * 2 * 3 \dots * N$$

Dalam pemrograman konsep dari faktorial seperti berikut,

$$\begin{aligned} \text{faktorial}(N) &= N! \\ &= N * (N-1)! \\ &= N * (N-1) * (N-2)! \\ &= N * (N-1) * (N-2) \dots * 3 * 2 * 1 \end{aligned}$$

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
```

### 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 katakunci (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 ini :

```
def perkalian(a, b):
    "Mengalikan dua bilangan"
    z = x * y
    print "Nilai a =", a
    print "Nilai b =", b
    print "a* b =", c
    # program utama mulai di sini
    perkalian(5,3)
    print
    perkalian(b=4,a=2)
    Hasilnya:
    Nilai a = 5
    Nilai b = 3
    a* b = 15
    Nilai a = 2
```

```
Nilai b = 4
a * b = 8
```

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(4,a=2)`, maka Anda akan mendapatkan pesan kesalahan sbb :

```
Traceback (innermost last):
File "./listing8.py", line 13, in ?
perkalian(4,x=2)
TypeError: keyword parameter redefined
```

Hasil ini menunjukkan pada kita bahwa nama `a` sudah dipakai. Dengan melihat pada definisi fungsi yang telah dibuat, parameter pertama adalah `a` dan kedua adalah `b`. Jadi ketika kita panggil dengan menyebutkan parameter kedua sebagai `a` 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 :

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

```

## P3.2 Contoh Kasus

### Contoh Kasus 1

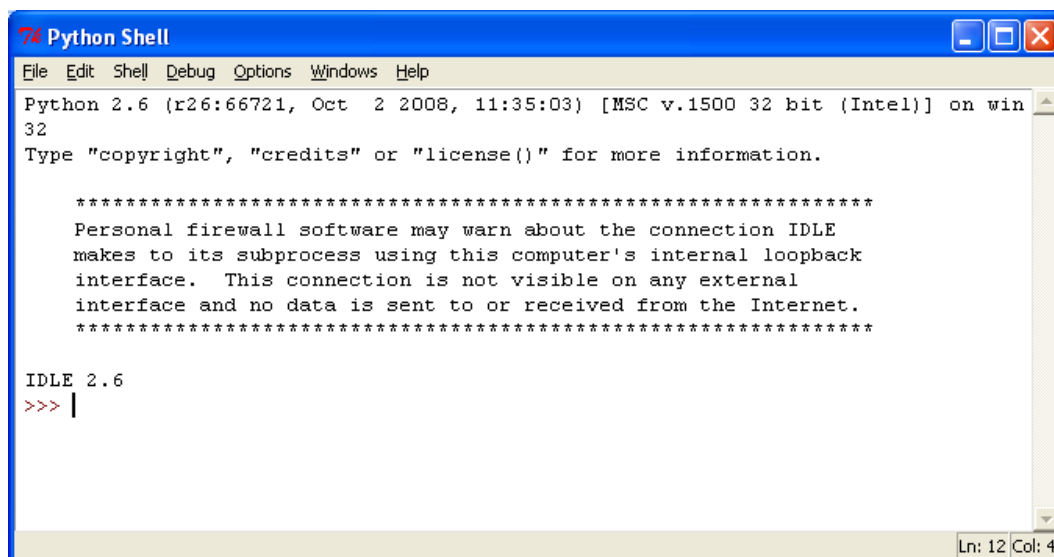
Pada contoh kasus yang pertama yaitu membuat program membuat fungsi perkalian dua bilangan bulat. Output yang akan ditampilkan adalah sebagai berikut :

```
IDLE 2.6
>>> ===== RESTART =====
>>>
Nilai x = 10
Nilai y = 2
x * y = 20

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

Langkah-langkah pengerjaan adalah sebagai berikut :

1. Klik tombol start Program Python26 IDLE(Python GUI), IDLE(GUI-Integrated Development Environment) dengan tampilan sebagai berikut :

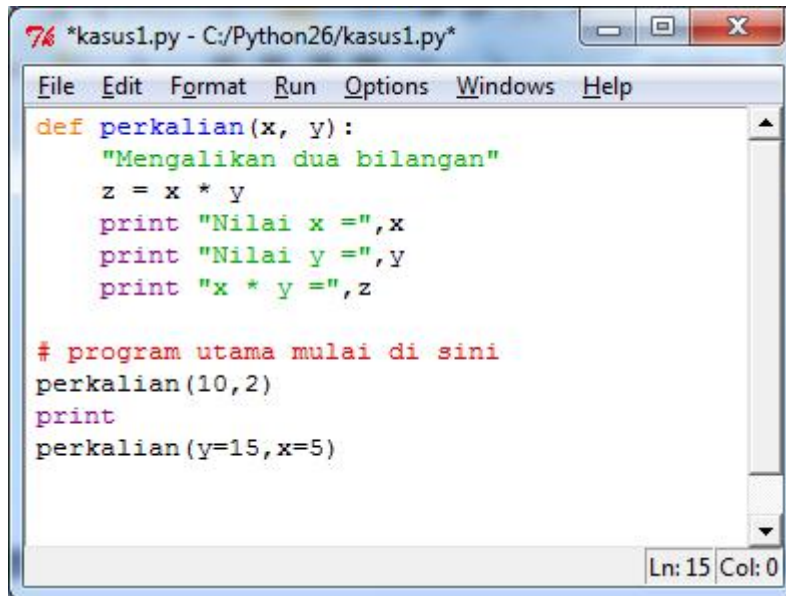


2. Klik Menu File -> New Window lalu ketikkan listing program sebagai berikut.

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

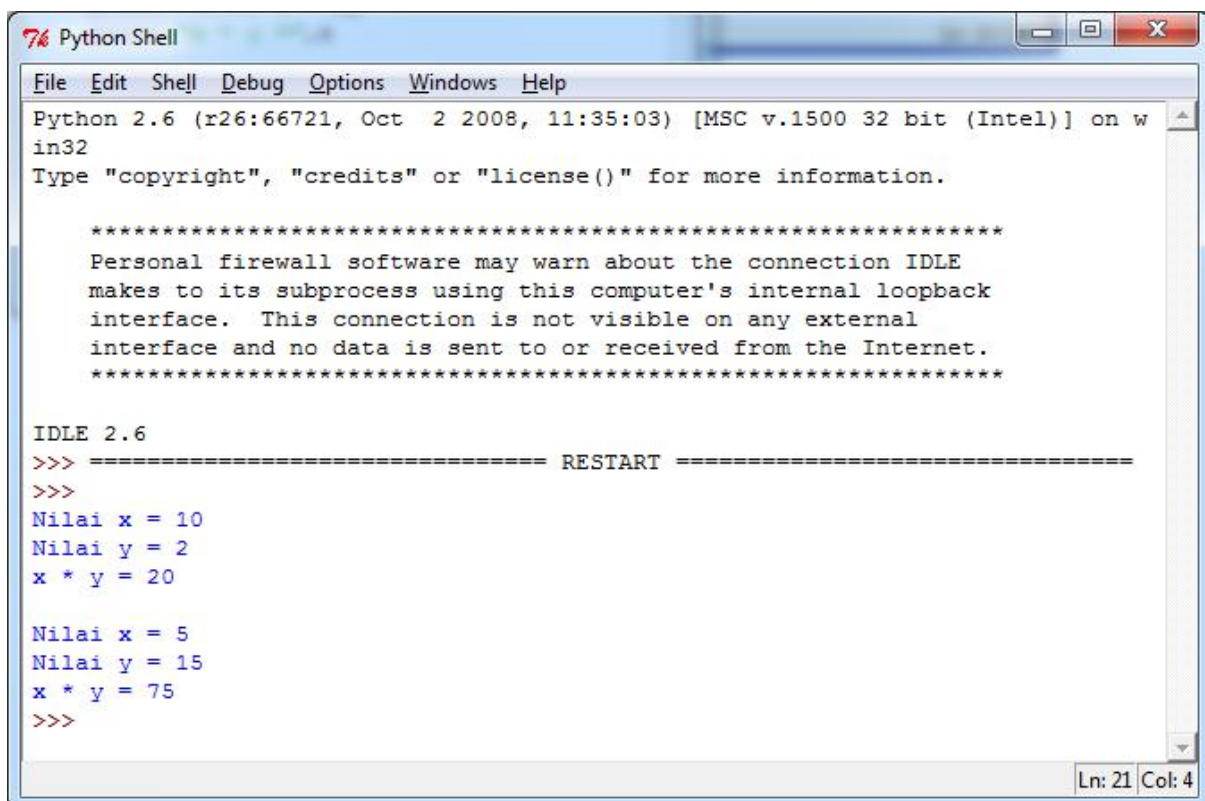
# program utama mulai di sini
perkalian(10,2)
print
perkalian(y=15,x=5)
```

- Setelah selesai mengetikkan code, langkah selanjutnya menyimpan file tersebut dengan cara klik menu File -> Save As. Masukkan nama file dengan nama kasus1.py



```
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)
```

- Setelah itu menjalankan program dengan cara klik menu Run -> Run Module atau dengan menekan tombol F5.



```
Python 2.6 (r26:66721, Oct 2 2008, 11:35:03) [MSC v.1500 32 bit (Intel)] on w  
in32  
Type "copyright", "credits" or "license()" for more information.  
  
*****  
Personal firewall software may warn about the connection IDLE  
makes to its subprocess using this computer's internal loopback  
interface. This connection is not visible on any external  
interface and no data is sent to or received from the Internet.  
*****  
  
IDLE 2.6  
>>> ===== RESTART =====  
>>>  
Nilai x = 10  
Nilai y = 2  
x * y = 20  
  
Nilai x = 5  
Nilai y = 15  
x * y = 75  
>>>
```

- Apabila tidak ada *error* maka program yang telah di-*compile* berhasil.



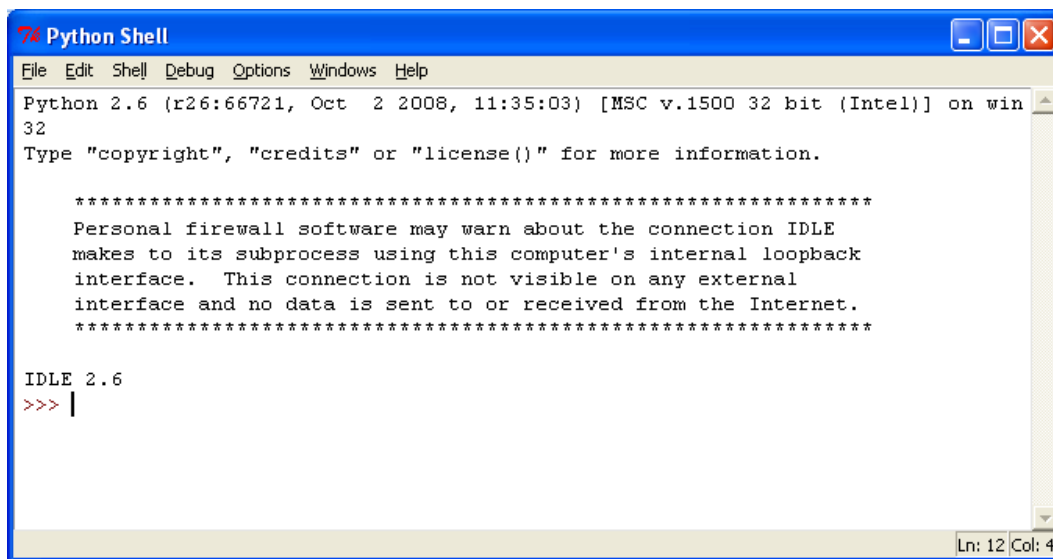
## Contoh Kasus 2

Pada contoh kasus yang kedua yaitu membuat program pencarian deret fibonacci dengan mengimplementasikan fungsi rekursif. Output yang akan ditampilkan adalah sebagai berikut :

```
IDLE 2.6      ==== No Subprocess ====
>>>
Masukkan sebuah bilangan : 25
Deret Fibonacci ke-(25) = 75025
>>> |
```

Langkah-langkah pengerjaan adalah sebagai berikut :

1. Klik tombol start Program Python26 IDLE(Python GUI), IDLE(GUI-Integrated Development Environment) dengan tampilan sebagai berikut :



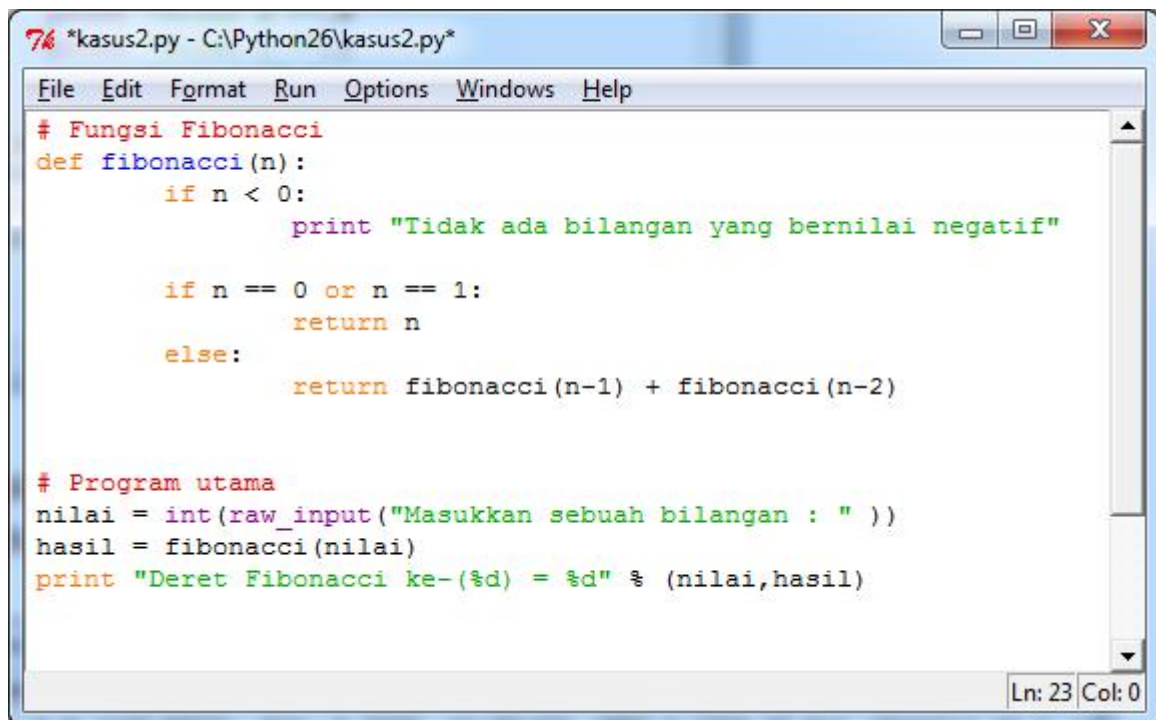
2. Klik Menu File -> New Window lalu ketikkan listing program sebagai berikut.

```
# 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 "Deret Fibonacci ke-(%d) = %d" % (nilai, hasil)
```

3. Setelah selesai mengetikkan code, langkah selanjutnya menyimpan file tersebut dengan cara klik menu File -> Save As. Masukkan nama file dengan nama kasus2.py

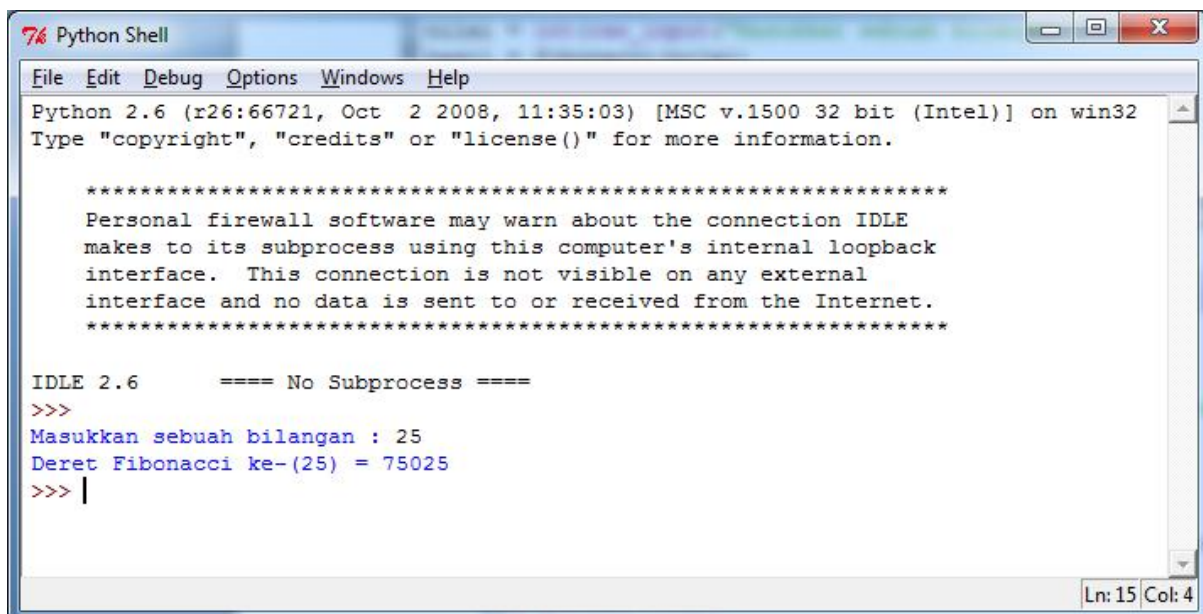


```
*kasus2.py - C:\Python26\kasus2.py*
File Edit Format Run Options Windows Help
# Fungsi Fibonacci
def fibonacci(n):
    if n < 0:
        print "Tidak ada bilangan yang bernilai negatif"

    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 "Deret Fibonacci ke-(%d) = %d" % (nilai, hasil)
Ln: 23 Col: 0
```

- Setelah itu menjalankan program dengan cara klik menu Run -> Run Module atau dengan menekan tombol F5.



```
Python Shell
File Edit Debug Options Windows Help
Python 2.6 (r26:66721, Oct 2 2008, 11:35:03) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6      ==== No Subprocess ====
>>>
Masukkan sebuah bilangan : 25
Deret Fibonacci ke-(25) = 75025
>>> |
Ln: 15 Col: 4
```

- Apabila tidak ada *error* maka program yang telah di-*compile* berhasil.

### P3.3 Latihan

Berikut ini merupakan program yang menggunakan bahasa python versi 2.6 untuk membuat menu dan melakukan perhitungan luas untuk tiga bangun ruang ( persegi panjang, lingkaran, dan segitiga). Pada code editor di python 2.6 ketikkan program berikut.

(Lengkapi kode program berikut dengan mengisi titik-titik yang berwarna merah)

```
#Mencetak Menu
.....
print "Menu Pilihan"
print
print "1. Persegi Panjang"
print "2. Lingkaran"
print "3. Segitiga"
print "4. Keluar"

.....
print "Menghitung Luas Persegi Panjang"
p = input("Masukkan Panjang : ") //pendeklarasian variabel input p
.....
.....
print "Luas Persegi Panjang adalah ",luas //menampilkan hasil variabel luas
print
print "Mau coba lagi [Y/N]? "
back = raw_input().upper()
if back == "Y":
    menu()
else:
    exit()

.....
print "Menghitung Luas Lingkaran"
r = input("Masukkan Jari-Jari : ")//pendeklarasian variabel input r
.....
.....
print
print "Mau coba lagi [Y/N]? "
back = raw_input().upper()
if back == "Y":
    menu()
else:
```

```

exit()

.....
print "Menghitung Luas Segitiga"
.....

.....
luas = (a*t)/2 //perhitungan untuk variabel luas
.....

print
print "Mau coba lagi [Y/N]? "
back = raw_input().upper()
if back == "Y":
    menu()
else:
    exit()

#Program Menghitung Luas
print "Selamat Datang di Program Untuk Menghitung Luas"
print "-----"
print
menu()

while 1:
    #input
    pilih = input("Masukkan pilihan : ")

    if pilih == 1:
        persegi()
        .....
        lingkaran()
        .....
        segitiga()
        .....
    print "\n"*100
    break
    else:
        print "Maaf pilihan yang anda masukkan tidak terdaftar"
        print "Coba lagi [Y/N] ? "
        coba = raw_input().upper()
        if coba == "Y":

```

```

menu()
else:
print "\n"*100
.....

```

(Save program diatas dengan nama file latihanfungsi.py)

## **TAMPILAN OUTPUT PROGRAM LATIHAN**

- **Tampilan Awal Menu**

```

IDLE 2.6
>>> ===== RESTART =====
=
>>>
Selamat Datang di Program Untuk Menghitung Luas
-----

Menu Pilihan

1. Persegi Panjang
2. Lingkaran
3. Segitiga
4. Keluar
Masukkan pilihan :

```

- **Tampilan Menu Persegi Panjang**

```

IDLE 2.6
>>> ===== RESTART =====
>>>
Selamat Datang di Program Untuk Menghitung Luas
-----

Menu Pilihan

1. Persegi Panjang
2. Lingkaran
3. Segitiga
4. Keluar
Masukkan pilihan : 1
Menghitung Luas Persegi Panjang
Masukkan Panjang : 10
Masukkan Lebar : 20
Luas Persegi Panjang adalah 200

Mau coba lagi [Y/N]?
Y

```

- Tampilan Menu Lingkaran

```
Menu Pilihan

1. Persegi Panjang
2. Lingkaran
3. Segitiga
4. Keluar
Masukkan pilihan : 2
Menghitung Luas Lingkaran
Masukkan Jari-Jari : 19
Luas Lingkaran adalah 1133.54

Mau coba lagi [Y/N]?
Y
```

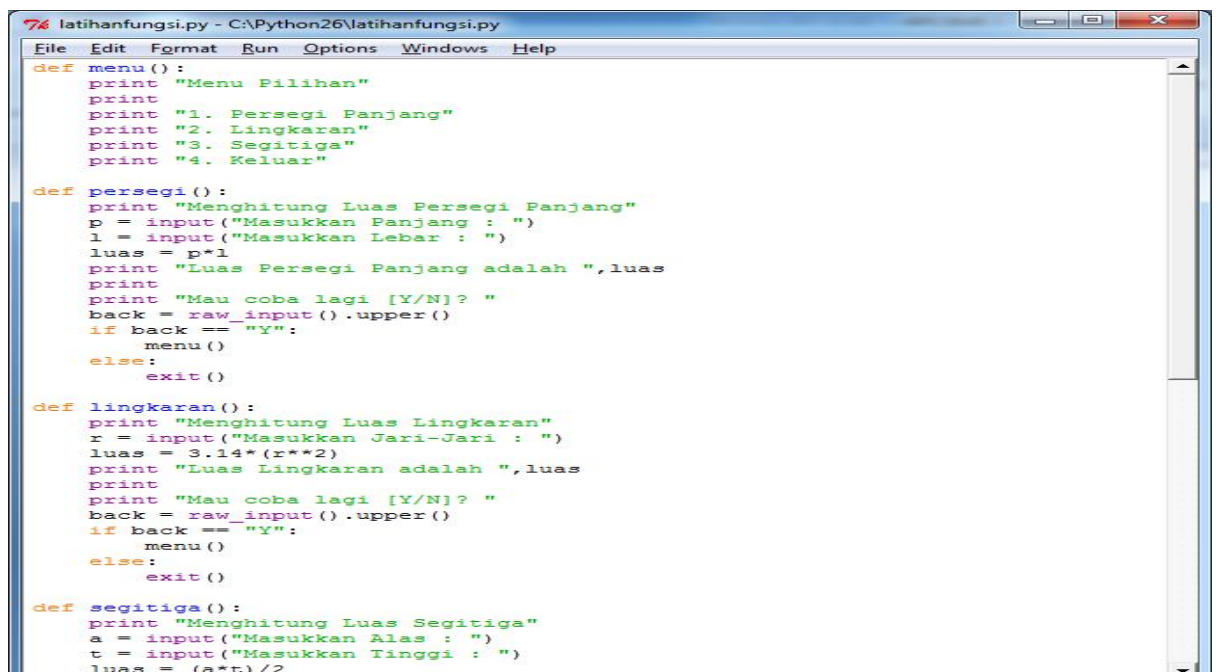
- Tampilan Menu Segitiga

```
Menu Pilihan

1. Persegi Panjang
2. Lingkaran
3. Segitiga
4. Keluar
Masukkan pilihan : 3
Menghitung Luas Segitiga
Masukkan Alas : 12
Masukkan Tinggi : 10
Luas Segitiga adalah 60

Mau coba lagi [Y/N]?
I
```

## KOREKSI PROGRAM LATIHAN



```
7% latihanfungsi.py - C:\Python26\latihanfungsi.py
File Edit Format Run Options Windows Help

def menu():
    print "Menu Pilihan"
    print
    print "1. Persegi Panjang"
    print "2. Lingkaran"
    print "3. Segitiga"
    print "4. Keluar"

def persegi():
    print "Menghitung Luas Persegi Panjang"
    p = input("Masukkan Panjang : ")
    l = input("Masukkan Lebar : ")
    luas = p*l
    print "Luas Persegi Panjang adalah ",luas
    print
    print "Mau coba lagi [Y/N]? "
    back = raw_input().upper()
    if back == "Y":
        menu()
    else:
        exit()

def lingkaran():
    print "Menghitung Luas Lingkaran"
    r = input("Masukkan Jari-Jari : ")
    luas = 3.14*(r**2)
    print "Luas Lingkaran adalah ",luas
    print
    print "Mau coba lagi [Y/N]? "
    back = raw_input().upper()
    if back == "Y":
        menu()
    else:
        exit()

def segitiga():
    print "Menghitung Luas Segitiga"
    a = input("Masukkan Alas : ")
    t = input("Masukkan Tinggi : ")
    luas = (a*t)/2
```

```

a = input("Masukkan Alas : ")
t = input("Masukkan Tinggi : ")
luas = (a*t)/2
print "Luas Segitiga adalah ",luas
print
print "Mau coba lagi [Y/N]? "
back = raw_input().upper()
if back == "Y":
    menu()
else:
    exit()

#Program Menghitung Luas
print "Selamat Datang di Program Untuk Menghitung Luas"
print "-----"
print
menu()

while 1:
    #input
    pilih = input("Masukkan pilihan : ")
    if pilih == 1:
        persegi()
    elif pilih == 2:
        lingkaran()
    elif pilih == 3:
        segitiga()
    elif pilih == 4:
        print "\n"*100
    else:
        print "Maaf pilihan yang anda masukkan tidak terdaftar"
        print "Coba lagi [Y/N] ? "

coba = raw_input().upper()
if coba == "Y":
    menu()
else:
    print "\n"*100

```

Ln: 1 Col: 0

### P3.4 DaftarPustaka

- [1] [www.id.m.wikipedia.org/wiki/Python\(bahasa\\_pemrograman\)](http://www.id.m.wikipedia.org/wiki/Python(bahasa_pemrograman)).
- [2] <http://www.master.web.id/mwmag/issue/03/content/tutorial-python-2/tutorial-python-2.html>
- [3] <http://dini3asa.staff.gunadarma.ac.id/Downloads/files/19688/FUNGSI.pdf>