Python Functions

Fungsi adalah blok kode terorganisir dan dapat digunakan kembali yang digunakan untuk melakukan tindakan tunggal dan terkait. Fungsi menyediakan modularitas yang lebih baik untuk aplikasi Anda dan tingkat penggunaan kode yang tinggi. Seperti yang sudah Anda ketahui, Python memberi Anda banyak fungsi built-in seperti cetak (), dll. Tetapi Anda juga dapat membuat fungsi Anda sendiri. Fungsi ini disebut fungsi yang ditentukan pengguna.

Defining a Function

Anda dapat menentukan fungsi untuk menyediakan fungsionalitas yang dibutuhkan. Berikut adalah aturan sederhana untuk mendefinisikan fungsi dengan Python.

• Blok fungsi dimulai dengan defensi kata kunci diikuti oleh nama fungsi dan tanda kurung (()).

• Setiap parameter masukan atau argumen harus ditempatkan di dalam tanda kurung ini. Anda juga dapat menentukan parameter di dalam tanda kurung ini.

• Pernyataan fungsi pertama dapat berupa pernyataan opsional - string dokumentasi fungsi atau docstring.

• Blok kode dalam setiap fungsi dimulai dengan titik dua (:) dan indentasi.

• Pernyataan kembali [ekspresi] keluar dari sebuah fungsi, secara opsional menyampaikan kembali ekspresi ke pemanggil. Pernyataan pengembalian tanpa argumen sama dengan return None.

Syntax

```
def functionname( parameters ):
        "function _docstring"
        function _suite
        return [expression]
```

Secara default, parameter memiliki perilaku posisi dan Anda perlu memberi tahu mereka dengan urutan yang sama seperti yang ditetapkan.

Example

Fungsi berikut mengambil string sebagai parameter masukan dan mencetaknya di layar standar.

```
def printme( str ):
        "This prints a passed string into this function"
        print str
        return
```

Calling a Function

Mendefinisikan sebuah fungsi hanya memberinya sebuah nama, menentukan parameter yang akan disertakan dalam fungsi dan menyusun blok kode. Setelah struktur dasar fungsi selesai, Anda dapat menjalankannya dengan memanggilnya dari fungsi lain atau langsung dari prompt Python. Berikut adalah contoh untuk memanggil fungsi printme () -

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
        "This prints a passed string into this function"
  print str
  return;

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
I'm first call to user defined function!
Again second call to the same function
```

Pass by reference vs value

Semua parameter (argumen) dalam bahasa Python dilewatkan dengan referensi. Ini berarti jika Anda mengubah parameter yang mengacu pada suatu fungsi, perubahan tersebut juga mencerminkan kembali fungsi pemanggilan. Sebagai contoh -

```
#!/usr/bin/python

# Function definition is here
def changeme( mylist ):
  "This changes a passed list into this function"
```

```
        mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return
        # Now you can call changeme function

mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Di sini, kita mempertahankan referensi objek yang dilewati dan menambahkan nilai pada objek yang sama. Jadi, ini akan menghasilkan hasil sebagai berikut -

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

Ada satu contoh lagi di mana argumen dilewatkan melalui referensi dan rujukannya ditimpa di dalam fungsi yang disebut.

```
#!/usr/bin/python

# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4]; # This would assig new reference in mylist
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Parameter mylist adalah local ke fungsi changeme. Mengubah mylist dalam fungsi tidak mempengaruhi mylist. Fungsi ini tidak menghasilkan apa-apa dan akhirnya ini akan menghasilkan hasil sebagai berikut:

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

Function Arguments

Anda dapat memanggil fungsi dengan menggunakan jenis argumen formal berikut:

- **Argumen yang dibutuhkan**
- **Argumen kata kunci**
- **Argumen baku**
- **Argumen panjang variable**

Required arguments

Argumen yang diperlukan adalah argumen yang diberikan ke sebuah fungsi dalam urutan posisi yang benar. Di sini, jumlah argumen dalam pemanggilan fungsi harus sesuai persis dengan definisi fungsi. Untuk memanggil fungsi printme (), Anda pasti perlu melewati satu argumen, jika tidak maka akan memberikan kesalahan sintaks sebagai berikut -

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
        "This prints a passed string into this function"
  print str
  return;

# Now you can call printme function
printme()
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut:

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Keyword arguments

Argumen kata kunci terkait dengan pemanggilan fungsi. Bila Anda menggunakan argumen kata kunci dalam pemanggilan fungsi, penelepon mengidentifikasi argumen berdasarkan nama parameter. Hal ini memungkinkan Anda melewatkan argumen atau menempatkannya agar tidak bermasalah karena penerjemah Python dapat menggunakan kata kunci yang diberikan agar sesuai dengan nilai parameter. Anda juga dapat membuat panggilan kata kunci ke fungsi printme () dengan cara berikut -

```
#!/usr/bin/python

# Function definition is here
```

```
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme( str = "My string")
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
My string
```

Contoh berikut memberikan gambaran yang lebih jelas. Perhatikan bahwa urutan parameter tidak masalah.

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Name:  miki
Age  50
```

Default arguments

Argumen default adalah argumen yang mengasumsikan nilai default jika nilai tidak diberikan dalam pemanggilan fungsi untuk argumen itu. Contoh berikut memberi ide pada argumen default, ini mencetak usia default jika tidak lulus -

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Variable-length arguments

Anda mungkin perlu memproses sebuah fungsi untuk argumen lebih banyak daripada yang Anda tentukan saat menentukan fungsinya. Argumen ini disebut variable-lengtharguments dan tidak disebutkan dalam definisi fungsi, tidak seperti argumen yang dibutuhkan dan standar.

Sintaks untuk fungsi dengan argumen variabel non-kata kunci adalah ini -

```
def functionname([formal _args,] *var _args _tuple ):
   "function _docstring"
   function _suite
   return [expression]
```

Tanda asterisk (*) ditempatkan sebelum nama variabel yang menyimpan nilai dari semua argumen variabel nonkeyword. Tuple ini tetap kosong jika tidak ada argumen tambahan yang ditentukan selama pemanggilan fungsi. Berikut adalah contoh sederhana -

```
#!/usr/bin/python

# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;
         # Now you can call printinfo function

printinfo( 10 )
printinfo( 70, 60, 50 )
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Output is:
10
Output is:
70
60
50
```

TheAnonymousFunctions

Fungsi ini disebut anonim karena tidak dinyatakan secara standar dengan menggunakan kata kunci def. Anda bisa menggunakan kata kunci lambda untuk membuat fungsi anonim yang kecil.

- **Bentuk lambda bisa mengambil sejumlah argumen tapi hanya mengembalikan satu nilai dalam bentuk ekspresi. Mereka tidak dapat berisi perintah atau beberapa ekspresi.**
- **Fungsi anonim tidak bisa menjadi panggilan langsung untuk dicetak karena lambda membutuhkan ekspresi**
- **Fungsi Lambda memiliki namespace lokal mereka sendiri dan tidak dapat mengakses variabel selain yang ada dalam daftar parameter dan yang ada di namespace global.**
- **Meskipun tampak bahwa lambda adalah versi satu baris dari sebuah fungsi, mereka tidak setara dengan pernyataan inline di C atau C ++, yang tujuannya adalah dengan melewatkan alokasi stack fungsi selama pemanggilan untuk alasan kinerja.**

Syntax

Sintaks fungsi lambda hanya berisi satu pernyataan, yaitu sebagai berikut -

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Berikut adalah contoh untuk menunjukkan bagaimana lambda bentuk fungsi bekerja -

```
#!/usr/bin/python

# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;


# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Value of total :  30
Value of total :  40
```

ThereturnStatement

Pernyataan kembali [ekspresi] keluar dari sebuah fungsi, secara opsional menyampaikan kembali ekspresi ke pemanggil. Pernyataan pengembalian tanpa argumen sama dengan return None.

Semua contoh di atas tidak mengembalikan nilai apapun. Anda bisa mengembalikan nilai dari sebuah fungsi sebagai berikut -

```
#!/usr/bin/python

# Function definition is here
def sum( arg1, arg2 ):
  # Add both the parameters and return them."
  total = arg1 + arg2
  print "Inside the function : ", total
  return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Inside the function :  30
Outside the function :  30
```

Scope of Variables

Semua variabel dalam sebuah program mungkin tidak dapat diakses di semua lokasi dalam program tersebut. Ini tergantung di mana Anda telah menyatakan sebuah variabel.

Ruang lingkup variabel menentukan bagian dari program di mana Anda dapat mengakses pengenal tertentu. Ada dua lingkup dasar variabel dengan Python -

- **Variabel global**
- **Variabel local**

Global vs. Local variables

Variabel yang didefinisikan di dalam badan fungsi memiliki lingkup lokal, dan yang didefinisikan di luar memiliki cakupan global.

Ini berarti bahwa variabel lokal dapat diakses hanya di dalam fungsi di mana mereka dideklarasikan, sedangkan variabel global dapat diakses di seluruh tubuh program oleh semua fungsi. Saat Anda memanggil fungsi, variabel yang dideklarasikan di dalamnya dibawa ke lingkup. Berikut adalah contoh sederhana -

```
#!/usr/bin/python

total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
  # Add both the parameters and return them."
  total = arg1 + arg2; # Here total is local variable.
  print "Inside the function local total : ", total
  return total;
        # Now you can call sum function

sum( 10, 20 );
print "Outside the function global total : ", total
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Inside the function local total :  30
Outside the function global total :  0
```