Fungsi - Definisi, Memanggil, dan Mengembalikan Fungsi

Fungsi

Di matematika, fungsi adalah proses yang merelasikan antara sebuah masukan (input) dan keluaran (output). Pada Python, selain fungsi relasi tersebut, fungsi juga adalah salah satu cara untuk mengorganisasikan kode - dengan tujuan akhir kode dapat digunakan kembali (*reusability*).

Beberapa syarat umum fungsi adalah modularisasi dan fungsionalitasnya. Jadi sebaiknya fungsi hanya memiliki satu kegunaan spesifik namun dapat digunakan kembali. Fungsi-fungsi umum telah disediakan oleh Python misalnya print(). Namun Anda dapat selalu mendefinisikan fungsi Anda sendiri.

Mendefinisikan fungsi

Fungsi didefinisikan dengan keyword def diikuti dengan nama fungsi dan parameternya dalam kurung ().

```
1. def nama_fungsi( parameter )
```

Secara opsional, Anda dapat menambahkan docstring - string dokumentasi yang menjelaskan konteks fungsi. Blok kode dalam setiap fungsi dimulai dengan titik dua dan menggunakan indentasi. Fungsi berhenti ketika terdapat statement return [expression] yang mengembalikan [expression] kepada pemanggilnya. Anda juga bisa membuat fungsi tidak mengembalikan keluaran dengan return None.

Sintaksis fungsi secara lengkap pada Python:

```
1. def nama_fungsi( parameter ):
2.     "dokumentasi fungsi"
3.     statemen atau kode fungsi
4.    return [expression]
```

Secara default, Python akan memposisikan setiap parameter sesuai dengan urutan pendaftaran pada saat didefinisikan, dan harus dipanggil sesuai dengan urutan tersebut. Contoh: fungsi berikut akan menerima sebuah string sebagai parameter dan mencetaknya.

```
    print(param1)
    return
```

Sintaksis return tanpa ekspresi atau return None dapat juga tidak dituliskan. Fungsi di atas akan sama seperti di bawah ini.

```
def cetak( param1 ):
    print(param1)
```

Memanggil Fungsi

Mendefinisikan sebuah fungsi hanya memberikan namanya, menentukan parameter yang ingin menjadi bagian dari fungsi dan struktur dasar kode tersebut. Setelah struktur dasar terpenuhi, Anda dapat memanggilnya pada fungsi yang lain atau dari Python prompt. Contoh berikut untuk memanggil fungsi cetak().

```
def cetak(param1):
    print(param1)
    return

#panggil
cetak("Panggilan Pertama")
cetak("Panggilan Kedua")
```

Saat kode diatas dijalankan, akan menghasilkan keluaran berikut:

Output:

```
Panggilan Pertama
Panggilan Kedua
```

Return

Pernyataan return [expression] akan membuat eksekusi program keluar dari fungsi saat itu, sekaligus mengembalikan nilai tertentu. Nilai return yang tidak mengembalikan (ekspresi) nilai bersifat sama dengan contoh di bawah ini.

```
return None
```

```
def kali(angka1, angka2):
    # Kalikan kedua parameter
    hasil = angka1 * angka2
    print('Dicetak dari dalam fungsi: {}'.format(hasil))
    return hasil

# Panggil fungsi kali
keluaran = kali(10, 20);
print('Dicetak sebagai kembalian: {}'.format(keluaran))
```

Saat dipanggil:

```
Dicetak dari dalam fungsi: 200

Dicetak sebagai kembalian: 200
```

Nilai kembalian dari sebuah fungsi dapat disimpan dalam sebuah variabel. Ini yang akan membedakan sebuah fungsi yang mengembalikan nilai dengan sebuah fungsi yang tidak mengembalikan nilai (sering disebut sebagai prosedur). Cobalah kode berikut ini:

```
def kuadrat(x):
    return x*x
a = 10
k = kuadrat(a)
print('nilai kuadrat dari {} adalah {}'.format(a, k))
```

Pass by reference vs value

Seluruh parameter (argumen) pada bahasa Python bersifat “passed by reference”. Artinya saat Anda mengubah sebuah variabel, maka data yang mereferensi padanya juga akan berubah, baik di dalam fungsi, maupun di luar fungsi pemanggil. **Kecuali jika anda melakukan operasi assignment yang akan mengubah reference parameter.**

Contohnya:

```
    list_saya.append([1, 2, 3, 4])
    print('Nilai di dalam fungsi: {}'.format(list_saya))

# Panggil fungsi ubah
list_saya = [10, 20, 30]
ubah(list_saya)
print('Nilai di luar fungsi: {}'.format(list_saya))
```

Dapat dilihat dalam kode diatas, objek list_saya yang direferensi adalah sama. Sehingga saat melakukan perubahan, maka perubahannya terjadi pada list_saya baik didalam maupun diluar fungsi ubah:

```
Nilai di dalam fungsi:  [10, 20, 30, [1, 2, 3, 4]]
Nilai di luar fungsi:  [10, 20, 30, [1, 2, 3, 4]]
```

Namun Anda harus berhati-hati karena assignment variabel bernama sama dengan parameter, berarti membuat variabel baru dalam scope lokal dan tidak terkait dengan variabel global.

```
def ubah(list_saya):
    "Deklarasi Variabel list_saya berikut hanya dikenali (berlaku) di dalam fungsi ubah"
    list_saya = [1, 2, 3, 4]
    print ('Nilai di dalam fungsi: {}'.format(list_saya))

# Panggil fungsi ubah
list_saya = [10, 20, 30]
ubah(list_saya)
print('Nilai di luar fungsi: {}'.format(list_saya))
```

Variabel list_saya dibuat kembali versi localnya dalam fungsi ubah dengan operator assignment (sama dengan), sehingga nilai list_saya akan berbeda karena bersifat lokal dalam fungsi ubah saja. Hasilnya akan sebagai berikut:

```
Nilai di dalam fungsi:  [1, 2, 3, 4]
Nilai di luar fungsi: [10, 20, 30]
```

> Tips: Untuk perubahan parameter sangat disarankan menggunakan keluaran fungsi

Argumen dan Parameter Fungsi

> Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what types of arguments a function can accept. For example, given the function definition:

Jadi parameter adalah definisi masukan yang diterima fungsi, dan argumen adalah hal yang Anda masukkan saat memanggil fungsi tersebut.

Contohnya, saat Anda membuat fungsi seperti berikut

```
1.  def fungsi_saya (a, b, c):
2.      #lakukansesuatudenganabc
```

maka a,b,c adalah parameter. Namun saat Anda memanggilnya

```
1.  fungsi_saya (1, b=14, c='Dicoding')
```

maka 1, 14, dan 'Dicoding' adalah argumen.

Sebelumnya

Selanjutnya