



# ALGORITMA PEMROGRAMAN

## 09. Procedure & Function

Fakultas Teknik  
Universitas Trunojoyo Madura

# 09. Function & Procedure

---

1. Definisi Procedure & Function
2. Struktur Dasar



Bukalapak  
@bukalapak

অনুসরণ করুন



Lelah.. Mungkin kata yang tepat untuk hari ini. Siapa yang udah nggak sabar sampai rumah dan rebahan di kasur?

Nah dari semua posisi tidur ini, mana yang jadi favorit kamu?

PS: Sekalian share sama kita divisi apa di kantor kamu yang jarang pulang ya :.)



### Posisi Tidur Pegawai Bukalapak



# 1. Definisi Function & Procedure

# a. Definisi Function & Procedure

- ❑ Sejumlah **pernyataan yang dikemas** dalam bentuk sub-program (potongan kode) yang bersifat **reuse**.
- ❑ Perbedaan mendasar antara prosedur dengan fungsi adalah ketidakhadiran suatu **“return”**
- ❑ Kalau fungsi mengembalikan suatu nilai setelah proses sedangkan prosedur tidak.
- ❑ Sebuah prosedur sama halnya dengan fungsi, ditulis dan kemudian dipanggil dengan menyebutkan namanya.
- ❑ Tujuan:
  - **Memudahkan** dalam mengembangkan program dengan membagi menjadi beberapa modul yang kecil.
  - **Menghemat ukuran & waktu** program.
  - program lebih **terstruktur**
  - **Reusability** oleh program atau fungsi lain.
  - Menghindari **penulisan perintah yang sama**

## 2. Struktur Dasar

- a. Struktur Dasar Function
- b. Pemanggilan Function
- c. Statemen Lambda
- d. Scope Variabel
- e. Rekursif

# a. Struktur Dasar

Tanpa Pengembalian Nilai	Dengan Pengembalian Nilai
<pre><b>def</b> nama_fungsi (arg1/param, ..., argN) :     perintah-perintah</pre>	<pre><b>def</b> nama_fungsi (arg1/param, ..., argN) :     perintah-perintah     <b>return</b> nilai_balik</pre>

1. Kata kunci **def** diikuti oleh **nama\_function** (nama fungsi), tanda kurung dan tanda **titik dua (:) menandai header (kepala) fungsi.**
2. **Parameter / argumen** adalah input dari luar yang akan diproses di dalam tubuh fungsi (variabel untuk menampung nilai untuk diproses di dalam fungsi).
4. Setelah itu diletakkan baris – baris pernyataan (**statements**). Jangan lupa **indentasi (tab/spasi)** untuk menandai blok fungsi.
5. **return** untuk mengembalikan suatu nilai **expression** dari fungsi.
  - Statemen *return* dan *parameter* sifatnya opsional, sesuai kebutuhan.

# a. Struktur Dasar

## Tanpa Pengembalian Nilai

```
# contoh dari prosedur
def tambah_prosedur(a, b):
    c = a + b
    print "Hasil Penjumlahan = ",c
```

## Dengan Pengembalian Nilai

```
# contoh dari fungsi
def tambah_fungsi(a, b):
    c = a + b
    return c
```

The diagram illustrates the components of a Python function definition and call. It includes the following code and annotations:

```
# Membuat Fungsi
def salam():
    print ("Hai, Selamat Pagi")
    print ("Selamat Datang")
## Pemanggilan Fungsi
salam()
```

Annotations with arrows pointing to the code:

- Kata kunci**: Points to the `def` keyword.
- Nama Function tanpa argumen ()**: Points to the function name `salam()` in the definition.
- Isi Function**: Points to the indented lines containing the function's body.
- Indentasi**: Points to the indentation of the function body lines.
- Pemanggilan Function**: Points to the `salam()` call at the bottom.



# a. Struktur Dasar

```
# Contoh Function
```

```
def kali(i,j):
```

```
    # mengalikan dua buah bilangan integer
```

```
    return i*j
```

```
x = 2
```

```
y = 3
```

```
hasil = kali(x,y)
```

```
print ("Ini Contoh Function")
```

```
print ("Hasil dari : ",x,'x',y, '=', hasil )
```

**Function**



```
# Contoh Procedure
```

```
def kali(i,j):
```

```
    # mengalikan dua buah bilangan integer
```

```
    print ("Ini Contoh Procedure")
```

```
    print ("hasil dari : ",x,'x',y, '=', i*j )
```

```
x = 2
```

```
y = 3
```

```
kali(x,y)
```

**Procedure**



## b. Pemanggilan Fungsi

- ❑ Memanggil fungsi tanpa parameter sebanyak 3x:

```
# Membuat Fungsi
def salam():
    print ("Hai, Selamat Datang")
## Pemanggilan Fungsi
salam()
salam()
salam()
```

```
Hai, Selamat Datang
Hai, Selamat Datang
Hai, Selamat Datang
```

- ❑ Memanggil fungsi dengan parameter

```
# Membuat Fungsi
def salam(cetak):
    print (cetak)
## Pemanggilan Fungsi
salam('Hai, Selamat Datang')
```


```
Hai, Selamat Datang
```

'Hai, Selamat Datang' adalah nilai parameter yang kita berikan.

## b. Pemanggilan Fungsi

Parameternya lebih dari satu, kita bisa menggunakan tanda koma (,) untuk memisahkannya.

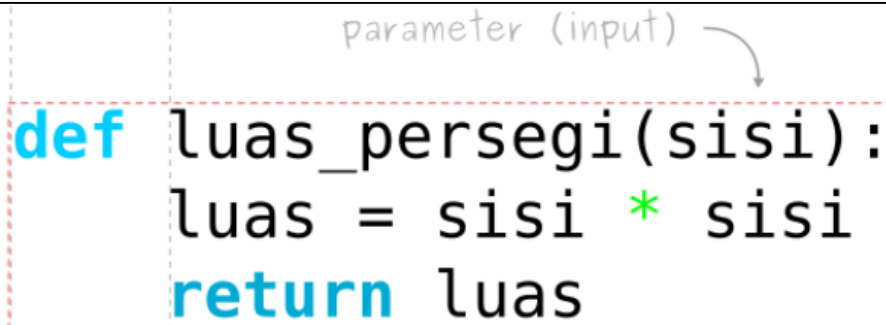
```
# Membuat fungsi dengan parameter
def luas_segitiga(alas, tinggi):
    luas = (alas * tinggi) / 2
    return luas
# Pemanggilan fungsi
print ("Luas segitiga: %f" % luas_segitiga(4, 6))
```



Parameter

## b. Pengembalian Fungsi (*return*)

Bila tidak ada pernyataan **return** ekspresi dikosongkan, maka fungsi akan mengembalikan objek **None**.



```
def luas_persegi(sisi):  
    luas = sisi * sisi  
    return luas
```

```
def Hitung_luas(p,l):  
    luas = p * l  
    return luas  
def Hitung_Volume(luas, t):  
    volume = luas * t  
    return volume  
def Hitung_Volume2(p,l,t):  
    volume = Hitung_luas(p,l) * t  
    return volume  
p = int(input("Masukkan Nilai panjang = "))  
l = int(input("Masukkan Nilai Lebar = "))  
t = int(input("Masukkan Nilai Tinggi = "))  
print("Luas = ", Hitung_luas(p,l))  
print("Volume = ", Hitung_Volume2(p,l,t))
```

```
Masukkan Nilai panjang = 20  
Masukkan Nilai Lebar = 23  
Masukkan Nilai Tinggi = 22  
Luas = 460  
Volume = 10120
```

```
# rumus: sisi x sisi  
def luas_persegi(sisi):  
    luas = sisi * sisi  
    return luas  
  
# rumus: sisi x sisi x sisi  
def volume_persegi(sisi):  
    volume = luas_persegi(sisi) * sisi  
    return volume  
  
# pemanggilan fungsi  
print("Luas persegi: %d" % luas_persegi(6))  
print("Volume persegi: %d" % volume_persegi(6))
```

## c. Statemen Lambda

### Struktur Dasarnya :

```
lambda arguments: expression
```

- ☐ Sebuah **ekspresi untuk membuat fungsi**.
- ☐ Disebut fungsi anonim karena fungsi ini tidak diberikan nama pada saat didefinisikan.
- ☐ Pada fungsi menggunakan kunci `def`, sedangkan fungsi anonim menggunakan kata kunci `lambda`.
- ☐ **Fungsi lambda** dapat mempunyai **banyak argumen**, tapi hanya **satu ekspresi**.
- ☐ **Ekspresi** tersebutlah yang **dikembalikan** sebagai **hasil fungsi**.
- ☐ Fungsi lambda dapat **disimpan** di dalam **variabel** untuk digunakan kemudian.
- ☐ Lambda biasanya **dibutuhkan** saat kita ingin **membuat fungsi dalam satu baris**.
- ☐ Biasanya saat menggunakan fungsi-fungsi seperti `filter()`, `map()`, dan `reduce()` kita akan membutuhkan lambda.

## c. Statemen Lambda

Kata kunci untuk  
membuat fungsi lambda

Argumen  
atau parameter

Isi fungsi

**lambda** **args** : **expression**

#Tanpa menggunakan lambda

```
def jumlah(a,b):
```

```
    c = a+b
```

```
    return c
```

```
hasil = jumlah(4,5)
```

```
print ("Tanpa menggunakan lambda",hasil)
```

# membuat anonymous function dengan lambda

```
kali = lambda x,y: x*y
```

```
hasil = kali(3,4)
```

```
print ("Menggunakan lambda",hasil)
```

```
=====
Tanpa menggunakan lambda 9
Menggunakan lambda 12
=====
```

## c. Statemen Lambda

```
# Filter bilangan ganjil dari list
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list_ganjil = list(filter(lambda x: x%2 != 0, my_list))
```

```
# Output: [1, 3, 5, 7, 9]
print(list_ganjil)
```

```
=====
[1, 3, 5, 7, 9]
=====
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
=====
```

```
# Program untuk menghasilkan kuadrat bilangan
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
kuadrat = list(map(lambda x: x*x, my_list))
```

```
# Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
print(kuadrat)
```

## d. Scope Variabel

- ❑ Di Python, tidak semua variabel bisa diakses dari semua tempat, tergantung dari tempat dimana mendefinisikan variabel.
- ❑ Saat menggunakan fungsi, ada namanya variabel **Global, Lokal dan Variabel Build-in** (sudah ada di python)
- ❑ Variabel Global adalah variabel yang bisa diakses dari semua fungsi,
- ❑ Variabel lokal hanya bisa diakses di dalam fungsi tempat ia berada saja.
- ❑ Urutan pengaksesan variabel (scope) disebut LGB (Local, Global, dan Build-in).



## d. Scope Variabel

```
total = 0
# Variabel global
# Definisi fungsi
def sum( arg1, arg2 ):
    total = arg1 + arg2;
    # total di sini adalah variabel lokal
    print ("Di dalam fungsi nilai total : ", total)
    return total
# Pemanggilan fungsi sum
sum( 10, 20 )
print ("Di luar fungsi, nilai total : ", total )
```

```
# membuat variabel global
nama = "Algoritma Pemrograman"
semester = "Satu"
def help():
    # ini variabel lokal
    nama = "Programan Berbasis Object"
    semester = "Tiga"
    # mengakses variabel lokal
    print ("Nama : %s" % nama)
    print ("semester : %s" % semester)
# mengakses variabel global
print ("Nama : %s" % nama)
print ("Semester : %s" % semester)
# memanggil fungsi help()
help()
```

```
=====
Di dalam fungsi nilai total : 30
Di luar fungsi, nilai total : 0
=====
```

```
=====
Nama : Algoritma Pemrograman
Semester : Satu
Nama : Programan Berbasis Object
semester : Tiga
=====
```

# d. Scope Variabel

## x,y variabel global - lokal

```
#variabel global
x="Saya"
y="Alpro"
#mendefinisikan fungsi ubah
def ubah():
    # variabel lokal
    x="Kamu"
    y='Siapa'
    print("Variabel Lokal : ",x,y)
    return x,y
#memanggil fungsi ubah
ubah()
#mencetak nilai x
print("Variabel Global : ",x,y)
```

```
=====
Variabel Lokal :  Kamu Siapa
Variabel Global :  Saya Alpro
=====
```

## x,y variabel global - global

```
#variabel global
x="Saya"
y="Alpro"
#mendefinisikan fungsi ubah
def ubah():
    # variabel lokal
    global x,y
    x="Kamu"
    y='Siapa'
    print("Variabel Lokal : ",x,y)
    return x,y
#memanggil fungsi ubah
ubah()
#mencetak nilai x
print("Variabel Global : ",x,y)
```

```
=====
Variabel Lokal :  Kamu Siapa
Variabel Global :  Kamu Siapa
=====
```

## e. Rekursif

- ❑ Adalah **function yang memanggil dirinya sendiri** secara langsung maupun tidak langsung melalui function yang lain.
- ❑ Rekursi fungsi berakhir saat bilangan sudah berkurang menjadi 1. disebut kondisi dasar (***base condition***).
- ❑ Setiap fungsi rekursi harus memiliki kondisi dasar, bila tidak, maka fungsi tidak akan pernah berhenti (***infinite loop***).
- ❑ **Keuntungan Menggunakan Rekursi**
  - Fungsi rekursi membuat kode terlihat lebih *clean* dan elegan
  - Fungsi yang rumit bisa dipecah menjadi lebih kecil dengan rekursi
  - Membangkitkan data berurut lebih mudah menggunakan rekursi ketimbang menggunakan iterasi bersarang.
- ❑ **Kerugian Menggunakan Rekursi**
  - Terkadang logika dibalik rekursi agak sukar untuk diikuti
  - Pemanggilan rekursi kurang efektif karena memakan lebih banyak memori
  - Fungsi rekursi lebih sulit untuk *didebug*

## e. Rekursif

```
# Contoh fungsi rekursi
# Menentukan faktorial dari bilangan
def faktorial(x):
    if x == 1:
        return 1
    else:
        return (x * faktorial(x-1))
bil = int(input("Masukan Bilangan : "))
print("Faktorial dari ",bil,"adalah :", faktorial(bil))
```

```
#Fungsi Rekursif untuk Menghitung Bilangan Berpangkat
def pangkat(x,y):
    if y == 0:
        return 1
    else:
        return x * pangkat(x,y-1)
x = int(input("Masukan Nilai X : "))
y = int(input("Masukan Nilai Y : "))
print("%d dipangkatkan %d = %d" % (x,y,pangkat(x,y)))
```

```
=====
Masukan Bilangan : 4
Faktorial dari 4 adalah : 24
=====
Masukan Nilai X : 2
Masukan Nilai Y : 2
2 dipangkatkan 2 = 4
=====
```