# Fungsi di Python

**Selamat Datang!** Notebook ini akan mengajarkan Anda tentang fungsi pada bahasa pemrograman Python. Akhir dari nnotebook ini, Anda akan tahu konsep dasar dari fungsi, variabel, dan bagaimana cara menggunakan fungsi.

## Daftar Isi



Prakiraan waktu yang dibutuhkan: **40 menit**

---

## Fungsi

Sebuah fungsi adalah kode yang dapat digunakan kembali yang mana operasinya ditentukan dalam fungsi. Fungsi membolehkan Anda memecah tugas dan menggunakan kembali kode dalam program yang berbeda.

Ada dua jenis fungsi :

- **Pre-defined Function**
- **User defined Function**

### Apa itu fungsi?

Anda dapat mendefinisikan fungsi untuk menyediakan fungsional yang dibutuhkan. Berikut adalah aturan sederhana dalam mendefinisikan sebuah fungsi dalam Python:

- Sebuah fungsi diawali `def` diikuti dengan `nama` fungsinya dan tanda kurung `()`.
- There are input parameters or arguments that should be placed within these parentheses.
- Anda juga bisa mendefinisikan parameter didalam tanda kurung ini
- Ada body didalam tiap fungsi yang dimulai dengan titik dua ( `:` ) dan kemudian kode dalam inden.
- Anda juga bisa menaruh dokumentasi sebelum body.
- Pernyataan `return` keluar dari fungsi, secara opsional meneruskan kembali suatu nilai.

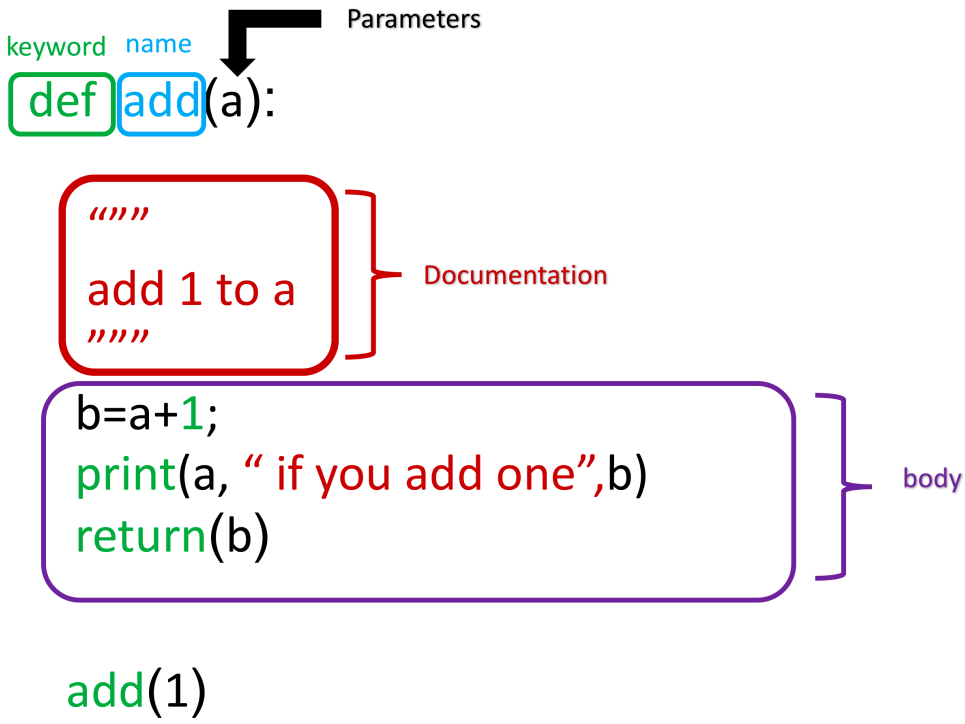
Sebagai contoh dari sebuah fungsi adalah add untuk menambahkan parameter `a` dan mengembalikan nilai `b` sebagai output:

In [1]:

```
# Contoh fungsi pertama: menambahkan 1 ke a dan menyimpannya sebagai b

def add(a):
    b = a + 1
    print(a, "if you add one", b)
    return(b)
```

Gambar berikut mengilustrasikan terminologinya:




Kita bisa mendapatkan bantuan tentang fungsi:

In [2]:

```
# Mendapatkan bantuan tentang fungsi add()

help(add)
```

```
Help on function add in module __main__:

add(a)
```

Memanggil fungsi:

In [3]:

```
# Memanggil fungsi add()

add(1)
```

```
1 if you add one 2
```

Out[3]:

```
2
```

Jika kita panggil fungsi dengan sebuah input baru, kita dapatkan hasil yang baru:

In [4]:

```
# Panggil fungsi add()

add(2)
```

```
2 if you add one 3
```

Out[4]:

```
3
```

Kita bisa membuat fungsi yang berbeda. Sebagai contoh, kita bisa membuat sebuah fungsi yang mengkalikan dua angka. Angka akan di representasikan oleh variabel `a` dan `b` :

In [5]:

```
# Mendefinisikan fungsi untuk mengkalikan dua buah angka

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

Fungsi yang sama bisa digunakan untuk tipe data yang berbeda. COntoohnya, kita akan mengkalikan dua integer:

In [6]:

```
# Gunakan mult() untuk mengkalikan dua integer

Mult(2, 3)
```

Out[6]:

```
6
```

Dua tipe data floats:

In [7]:

```
# Gunakanan mult() untuk mengkalikan dua tipe data floats

Mult(10.0, 3.14)
```

Out[7]:

```
31.400000000000002
```

Kita bahkan bisa mereplikasi string dengan mengkalikannya dengan bilangan bulat:

In [8]:

```
# Gunakan mult() untuk mengkalikan dua tipe data berbeda secara bersamaan

Mult(2, "Michael Jackson ")
```

Out[8]:

```
'Michael Jackson Michael Jackson '
```

## Variabel

Input ke dalam sebuah fungsi disebut dengan formal parameter.

Sebuah variabel yang dideklarasikan didalam fungsi disebut dengan variabel lokal. Parameter hanya ada didalam fungsi (yaitu dimana fungsi dimulai dan berhenti)

Sebuah variabel yang dideklarasikan diluar fungsi adalah variabel global, dan nilainya dapat diakses dan di modifikasi melalui program. Kita akan membahasnya lebih lanjut tentang variabel global di akhir.

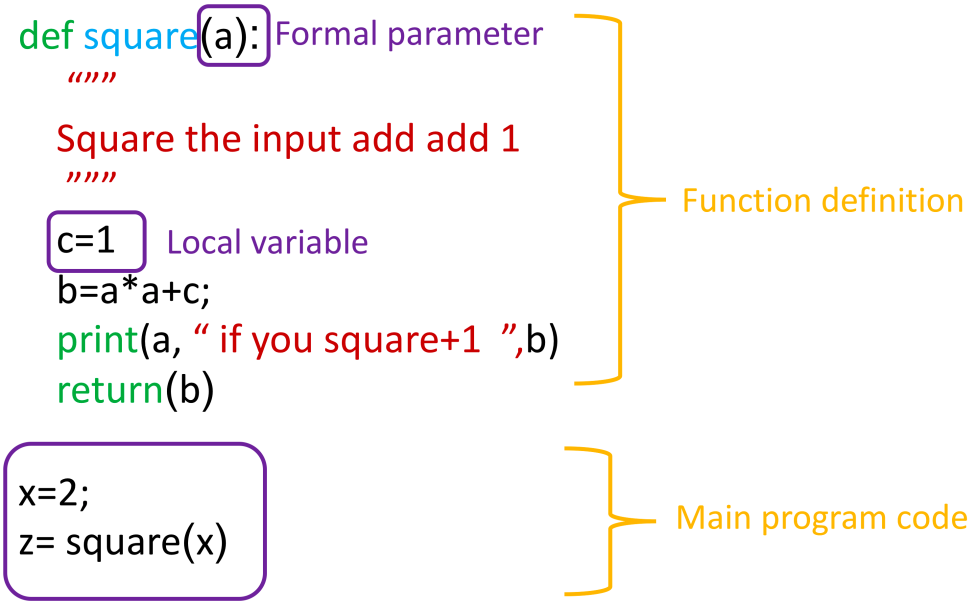
In [9]:

```
# Mendefinisikan fungsi

def square(a):

    # Local variable b
    b = 1
    c = a * a + b
    print(a, "if you square + 1", c)
    return(c)
```

Label ditampilkan pada gambar dibawah:




Kita dapat memanggil fungsi dengan input **3** :

In [10]:

```
# Inisialisasi variabel global

x = 3
# Buat fungsi memanggil dan mengembalikan fungsi y
y = square(x)
y
```

```
3 if you square + 1 10
```

Out[10]:

```
10
```

Kita dapat memanggi fungsi dengan input **2** dengan cara yang lain:

In [11]:

```
#Secara langsung masukkan angka sebagai parameter

square(2)
```

```
2 if you square + 1 5
```

Out[11]:

```
5
```

Jika disana tidak ada pernyataan `return` , fungsi tersebut akan mengembalikan `None` . Berikut adalah dua fungsi yang mirip:

In [12]:

```
# Definisikan fungsi, satu mengembailkan nilai None dan lainnya tanpa nilai yang dikemb
alikan

def MJ():
    print('Michael Jackson')

def MJ1():
    print('Michael Jackson')
    return(None)
```

In [13]:

```
# Lihat output-nya

MJ()
```

```
Michael Jackson
```

In [14]:

```
# Lihat output-nya

MJ1()
```

```
Michael Jackson
```

Dengan print sebuah fungsi dapat mengetahui apakah mengembalikan **None** sebagai nilai returnnya:

In [15]:

```
# Lihat apa nilai yang dikembalikan oleh fungsi

print(MJ())
print(MJ1())
```

```
Michael Jackson
None
Michael Jackson
None
```

Buat sebuah fungsi `con` that menggabungkan dua string dengan menggunakan operasi tambahan:

In [16]:

```
# Definisikan fungsi untuk menggabungkan string

def con(a, b):
    return(a + b)
```

In [17]:

```
# Tes pada fungsi con()

con("This ", "is")
```

Out[17]:

```
'This is'
```

---

**[Tip] Bagaimana Saya mempelajari lebih tentang pre-defined function pada Python?**

Kita akan mengenalkan variasi dari pre-defined functions. Ada banyak fungsi, jadi tidak mungkin kita mengajarkan semuanya dalam sekali kelas. Tapi jika Anda bersedia untuk belajar cepat, berikut ada referensi singkat untuk beberapa pre-defined functions: Reference (http://www.astro.up.pt/~sousasag/Python_For_Astronomers/Python_qr.pdf)

---

## Fungsi membuat segalanya sederhana

Perhatikan dua baris kode **Block 1** dan **Block 2**: prosedur untuk tiap barisnya memiliki kemiripan. Satu hal yang berbeda adalah nama variabel dan nilainya.

**Block 1:**

In [18]:

```
# block 1 kalkulasi a dan b

a1 = 4
b1 = 5
c1 = a1 + b1 + 2 * a1 * b1 - 1
if(c1 < 0):
    c1 = 0
else:
    c1 = 5
c1
```

Out[18]:

```
5
```

**Block 2:**

In [19]:

```
# block 2 kalkulasi a dan b

a2 = 0
b2 = 0
c2 = a2 + b2 + 2 * a2 * b2 - 1
if(c2 < 0):
    c2 = 0
else:
    c2 = 5
c2
```

Out[19]:

```
0
```

Kita dapat mengganti baris kode dengan sebuah fungsi. Sebuah fungsi menggabungkan banyak instruksi menjadi satu baris kode. Sekali fungsi didefinisikan, itu dapat digunakan secara berulang-ulang. Anda dapat meminta fungsi yang sama beberapa kali dalam program Anda. Anda dapat menghemat fungsi dan menggunakannya pada program yang lain atau gunakan fungsi lainnya. Baris kode pada **Block 1** dan **Block 2** dapat digantikan dengan fungsi berikut:

In [20]:

```
# Buat fungsi untuk mengkalkulasi diatas

def Equation(a,b):
    c = a + b + 2 * a * b - 1
    if(c < 0):
        c = 0
    else:
        c = 5
    return(c)
```

Fungsi ini mengambil dua input, a dan b, kemudian menerapkan beberapa operasi untuk mengembalikan nilai c. Dengan sederhana kita definisikan fungsi, mengganti instruksi dengan fungsi, dan masukkan nilai yang baru dari `a1`, `b1` dan `a2`, `b2` sebagai input. Proses tersebut didemonstrasikan pada gambar:

```
a1=5
b1=5
c1=a1+b1+2*a1*b1-1
   if(c1<0):
      c1=0
   else:
      c1=5
```

Kode **Blocks 1** dan **Block 2** sekarang dapat digantikan dengan**Block 3** dan **Block 4**.

**Block 3:**

In [21]:

```
a1 = 4
b1 = 5
c1 = Equation(a1, b1)
c1
```

Out[21]:

5

**Block 4:**

In [22]:

```
a2 = 0
b2 = 0
c2 = Equation(a2, b2)
c2
```

Out[22]:

```
0
```

---

# Pre-defined functions

Ada banyak pre-defined function pada Python, mari kita mulai dengan yang sederhana dahulu.

Fungsi `print()` :

In [23]:

```
# Fungsi print() bawaan

album_ratings = [10.0, 8.5, 9.5, 7.0, 7.0, 9.5, 9.0, 9.5]
print(album_ratings)
```

```
[10.0, 8.5, 9.5, 7.0, 7.0, 9.5, 9.0, 9.5]
```

Fungsi `sum()` menjumlahkan semua elemen dalam list atau tuple

In [24]:

```
# Gunakan sum() untuk menambahkan setiap elemen dalam list atau tuple

sum(album_ratings)
```

Out[24]:

```
70.0
```

Fungsi `len()` mengembalikan panjang dari lis atau tuple

In [25]:

```
# Tampilkan panjang dari list atau tuple

len(album_ratings)
```

Out[25]:

```
8
```

# Menggunakan pernyataan `if/else` dan Loops dalam Fungsi

Fungsi `return()` berguna terutama jika Anda memiliki beberapa pernyataan IF dalam fungsi, ketika Anda ingin output Andae menjadi tergantung (dependent) pada beberapa kondisi:

In [26]:

```
# Contoh fungsi

def type_of_album(artist, album, year_released):

    print(artist, album, year_released)
    if year_released > 1980:
        return "Modern"
    else:
        return "Oldie"

x = type_of_album("Michael Jackson", "Thriller", 1980)
print(x)
```

```
Michael Jackson Thriller 1980
Oldie
```

Kita dapat menggunakan perulangan dalam sebuah fungsi. Contohnya, kita bisa `print` tiap elemen dalam list:

In [27]:

```
# Print list menggunakan perulangan

def PrintList(the_list):
    for element in the_list:
        print(element)
```

In [28]:

```
# Menerapkan fungsi printlist

PrintList(['1', 1, 'the man', "abc"])
```

```
1
1
the man
abc
```

# Mengatur nilai argumen standar pada fungsi Anda

Anda dapat mengatur argumen default pada fungsi Anda. Contohnya, pada fungsi `isGoodRating()`, bagaimana jika kita ingin membuat threshold untuk apa yang kita tentukan menjadi good rating ? Mungkin secara default, kita memiliki rating defaultnya adalah 4:

In [29]:

```
# Contoh untuk mengatur parameter dengan nilai default

def isGoodRating(rating=4):
    if(rating < 7):
        print("this album sucks it's rating is",rating)

    else:
        print("this album is good its rating is",rating)
```

In [30]:

```
# Uji nilainya dengan nilai default dan dengan nilai input

isGoodRating()
isGoodRating(10)
```

```
this album sucks it's rating is 4
this album is good its rating is 10
```

---

# Variabel Global

Sejauh ini kita telah membuat variabel dalam fungsi, tapi kita belum mendiskusikan variabel diluar fungsi. Ini disebut dengan variabel global.
Let's try to see what `printer1` returns:

In [31]:

```
# Contoh dari variabel global

artist = "Michael Jackson"
def printer1(artist):
    internal_var = artist
    print(artist, "is an artist")

printer1(artist)
print(internal_var)
```

```
Michael Jackson is an artist

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-31-dd8d4a07b7e2> in <module>()
      7
      8 printer1(artist)
----> 9 print(internal_var)

NameError: name 'internal_var' is not defined
```

Jika kita lalukan perintah print pada `internal_var` , kita mendapatkan error.

**Kita dapatkan NameError: `name 'internal_var' is not defined` . Mengapa?**

Itu dikarenakan semua variabel yang kita buat dalam fungsi adalah **variabel lokal**, yang artinya variabel tersebut tidak ada diluar fungsi.

Tapi ada cara untuk membuat **variabel global** dari dalam fungsi seperti berikut:

In [ ]:

```
artist = "Michael Jackson"

def printer(artist):
    global internal_var
    internal_var= "Whitney Houston"
    print(artist,"is an artist")

printer(artist)
printer(internal_var)
```

# Ruang Lingkup Variabel

Ruang lingkup dari variabel adalah bagian dari program dimana variabel dapat diakses. Variabel dideklarasikan diluar fungsi, seperti variabel `myFavouriteBand` pada kode yang ditunjukkan disini adalah dapat diakses dari mana saja didalam program. Hasilnya, beberapa variabel dikatakan memiliki ruang lingkup global, dan diketahui sebagai variabel global. `myFavouriteBand` adalah variabel global, jadi bisa diakses dari dalam fungsi `getBandRating` , dan kita dapat gunakan itu untuk menentukan rating dari band. Kita juga dapat gunakan itu diluar fungsi, seperti ketika kita lakukan print fungsi untuk menampilkannya:

In [ ]:

```
# Contoh dari variabel global

myFavouriteBand = "AC/DC"

def getBandRating(bandname):
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is:", getBandRating("AC/DC"))
print("Deep Purple's rating is:",getBandRating("Deep Purple"))
print("My favourite band is:", myFavouriteBand)
```

Lihat versi dari kode yang telah dimodifikasi. Sekarang variabel `myFavouriteBand` didefinisikan dalam fungsi `getBandRating` . Sebuah variabel yang didefinisikan dalam fungsi dikatakan variabel lokal dari fungsi tersebut. Artinya variabel tersebut hanya bisa diakses dari dalam fungsi dimana itu didefinisikan. Fungsi `getBandRating` akan tetap bekerja, karena `myFavouriteBand` tetap didefinisikan dalam fungsi. Namun, kita tidak bisa lagi menampilkan `myFavouriteBand` diluar fungsi, karena itu adalah variabel lokal dari fungsi `getBandRating` ; ituhanya didefinisikan dalam fungsi `getBandRating` :

In [ ]:

```
# Contoh variabel lokal

def getBandRating(bandname):
    myFavouriteBand = "AC/DC"
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is: ", getBandRating("AC/DC"))
print("Deep Purple's rating is: ", getBandRating("Deep Purple"))
print("My favourite band is", myFavouriteBand)
```

Akhirnya, lihat pada contoh. Kita sekarang memiliki dua definisi variabel `myFavouriteBand`. Pertama adalah global, dan kedua adalah variabel lokal didalam fungsi `getBandRating`. Didalam fungsi `getBandRating`, variabel lokal memiliiki hak lebih tinggi. **Deep Purple** akan menerima rating 10.0 ketika melewati fungsi `getBandRating`. Namun, diluar dari fungsi `getBandRating`, variabel lokal `getBandRating` tidak didefinisikan, jadi variabel `myFavouriteBand` yang di print adalah variabel global, dimana memiliki nilai **AC/DC**.

In [ ]:

```
# Contoh dari variabel global dan lokal dengan nama yang sama

myFavouriteBand = "AC/DC"

def getBandRating(bandname):
    myFavouriteBand = "Deep Purple"
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is:",getBandRating("AC/DC"))
print("Deep Purple's rating is: ",getBandRating("Deep Purple"))
print("My favourite band is:",myFavouriteBand)
```

# Kuis mengenai Functions

Tampilkan fungsi yang membagi input pertama dengan input kedua:

In [7]:

```
# Tulis kode anda dibawah ini dan tekan Shift+Enter untuk melakukan eksekusi
def div  (a,b):
    return (a/b)
div (8,4)
```

Out[7]:

```
2.0
```

klik dua kali **disini** untuk melihat jawabannya.

Gunakan fungsi `con` untuk pertanyaan berikut.

In [9]:

```
# Gunakan fungsi con untuk pertanyaan berikut

def con(a, b):
    return(a * b)
con (15,30)
```

Out[9]:

```
450
```

Bisakah fungsi `con` kita definisikan sebelum digunakan untuk menambahkan integer atau string?

In [11]:

```
# Tulis kode anda dibawah ini dan tekan Shift+Enter untuk melakukan eksekusi
con (10,100)
```

Out[11]:

```
1000
```

klik dua kali **disini** untuk melihat jawabannya.

Bisakah fungsi `con` kita definisikan sebelum digunakan untuk menggabungkan sebuah list atau tuple ?

In [14]:

```
# Tulis kode anda dibawah ini dan tekan Shift+Enter untuk melakukan eksekusi
con(["coba", "test"], ["sebelum","definisi"])
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-14-cdaa0a6e907a> in <module>()
      1 # Tulis kode anda dibawah ini dan tekan Shift+Enter untuk melakuka
n eksekusi
----> 2 con(["coba", "test"], ["sebelum","definisi"])

<ipython-input-9-f787eeccb1ed> in con(a, b)
      2
      3 def con(a, b):
----> 4     return(a * b)
      5 con (15,30)

TypeError: can't multiply sequence by non-int of type 'list'
```

klik dua kali **disini** untuk melihat jawabannya.

## About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) is a Data Scientist at IBM, and holds a PhD in Electrical Engineering. His research focused on using Machine Learning, Signal Processing, and Computer Vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a), James Reeve (https://www.linkedin.com/in/reevejamesd/)

---