FUNGSI

(Part 2)

MELEWATKAN FUNGSI SEBAGAI ARGUMENT

Melewatkan Fungsi Sebagai Argument

- Pada pemrograman Python, kita dapat **melewatkan** suatu **fungsi** sebagai **argument** fungsi yang lainnya.
- Caranya seperti halnya melewatkan argument-argument yang berupa variabel.

Format/Syntax:

```
def nama_fungsi(fungsi1, fungsi2):
    statement1
    statement2
```

Contoh:

```
# Melewatkan fungsi sebagai argument
def plus():
    hasil=17+7
    return hasil

def minus():
    hasil=27-8
    return hasil

def tampilkan(a,b):
    print("Hasil penjumlahan adalah: ",plus())
    print("Hasil pengurangan adalah: ",minus())

def program_utama():
    tampilkan(plus,minus)

program_utama()
```

```
#Output
Hasil penjumlahan adalah:  24
Hasil pengurangan adalah:  19
```

FUNGSI SEBAGAI NILAI BALIK (RETURN)

Melewatkan Fungsi Sebagai Nilai Balik

- Kelebihan pemrograman Python yang lain adalah kita dapat membuat **fungsi** sebagai **nilai balik** (*Return value*) suatu fungsi.
- Caranya seperti halnya mengembalikan obyek/nilai yang berupa variabel biasa.

Format/Syntax:

```
def nama_fungsi(fungsi):
    statement1
    statement2
    return fungsi
```

Contoh:

```
# Melewatkan fungsi sebagai nilai balik
def Data():
    print("Selamat belajar Python!")

def Nilai(fungsi):
    print("Fungsi sebagai nilai balik")
    return fungsi

T = Nilai(Data)
T()
```

```
#Output
Fungsi sebagai nilai balik
Selamat belajar Python!
```

FUNGSI DI DALAM FUNGSI
(Nested Function)

Fungsi Bersarang (Nested)

- Kelebihan pemrograman Python selanjutnya adalah kita dapat membuat fungsi di dalam fungsi atau sering disebut dengan fungsi bersarang (*Nested*)

Contoh:

```
# Fungsi Bersarang (Nested)
def fungsi_A():                                     # Deklarasi fungsi_A
    def fungsi_B():                                 # Deklarasi fungsi_B
        print("Hi ... saya fungsi_B")
        print("Fungsi A memanggil Fungsi_B")
    fungsi_B()                                      # Memanggil fungsi_B

fungsi_A()                                          # Memanggil fungsi_A
```

```
#Output
Hi ... saya fungsi_B
Fungsi A memanggil Fungsi_B
```

DECORATOR
(Bentuk Lain Fungsi Bersarang)

Decorator

- **Decorator** merupakan cara lain dalam pemrograman Python untuk menangani **pembungkusan fungsi** atau fungsi bersarang (***Nested function***).
- Perbedaan **decorator** dengan fungsi bersarang adalah **pada saat pemanggilannya**.
- Pemanggilan decorator diawali dengan karakter "**@**" dan dikuti nama fungsi.
- Format/*Syntax* sebuah decorator, sebagai berikut:

```
@nama_fungsi
```

Contoh:

```
# Bentuk Lain Fungsi Bersarang (Nested) - Decorator

def myDecorator(t):
        def F_bersarang(a):                    # Fungsi bersarang/Nested
            print("Assalamu'alaikum")
            return t(a)                        # Mengembalikan alamat fungsi
        return F_bersarang

@myDecorator
def hello(b):
    print(b)                                    # hello=myDecorator(hello)

hello("Sahabat Robonesia")
```

```
# Output
Assalamu'alaikum
Sahabat Robonesia
```

```
@myDecorator
def hello(b):
    print(b)
```

Blok program ini artinya adalah **`hello = myDecorator(hello)`**

Catatan Decorator

- **Decorator** pada umumnya mengembalikan (*return*) suatu alamat fungsi.
- **Decorator** merupakan fungsi yang **memiliki argument** berupa **fungsi**.

OVERLOADING FUNGSI PADA PYTHON

Overloading Fungsi

- Overloading fungsi adalah suatu Teknik dalam pemrograman yang memungkinkan suatu fungsi **memiliki nama yang sama** dengan fungsi yang lainnya, namun dengan **argument/parameter** yang **berbeda**.
- Namun tidak seperti bahasa pemrograman C/C++, pada Python **tidak mengenal overloading fungsi**.

Contoh:

```
# Overloading fungsi

def salam():
    print("Assalamu'alaikum, saya Robonesia")

def salam(t):
    print("Assalamu'alaikum, saya adalah", t)

salam()
salam("Robonesia")
```

```
# Output
Traceback (most recent call last):
  File "E:\PROGRAMMING\Pemrograman Python - 2 -
PyCharm\pycharm_workspace\overloading.py", line 9, in <module>
    salam()
TypeError: salam() missing 1 required positional argument: 't'
```

REKURSI

Fungsi Rekursi

- **Rekursi** adalah teknik dalam pemrograman fungsional yang memungkinkan suatu **fungsi dapat memanggil dirinya sendiri**.
- Fungsi yang bersifat rekursif biasa disebut dengan **fungsi rekursi**.

Contoh:

```
# Fungsi rekursi

def display(t):
    if t>0:
        print("Belajar fungsi Rekursi")
        display(t-1)          # Fungsi memanggil diri sendiri (Rekursi)

def main():
    display(3)                # Memanggil fungsi rekursif dengan berbatas

main()              # Panggil fungsi utama
```

```
# Output
Belajar fungsi Rekursi
Belajar fungsi Rekursi
Belajar fungsi Rekursi
```

Kesalahan dalam Fungsi Rekursi

```
# Kesalahan dalam Fungsi rekursi

def display():
        print("Belajar fungsi Rekursi")
        display()    # Fungsi rekursif yang salah
def main():
    display()        # Memanggil fungsi rekursif tanpa batas

main()               # Panggil fungsi utama
```

```
# Output
Belajar fungsi Rekursi
Belajar fungsi Rekursi
Belajar fungsi Rekursi
...
Traceback (most recent call last):
  [Previous line repeated 992 more times]
  File "E:\PROGRAMMING\Pemrograman Python - 2 -
PyCharm\pycharm_workspace\fungsi_rekursi.py", line 17, in display
    print("Belajar fungsi Rekursi")
RecursionError: maximum recursion depth exceeded while calling a Python object
```

Pemecahan Masalah Matematika Menggunakan Rekursi

1. Rekursi & Faktorial

Faktorial suatu bilangan dapat diformulasikan sebagai berikut:

```
n! =1 x 2 x 3 x 4 x ... x n
```

atau

```
Faktorial(n) = n!
             = n*(n-1)!
             = n*(n-1)*(n-2)!
```

Contoh Rekursi & Faktorial:

```
# Fungsi rekursi & Faktorial

def Faktorial(t):                   # Fungsi faktorial
    if t==0:
        return 1
    else:
        return t*Faktorial(t-1)     # Rekursi fungsi faktorial

def main():
    t = int(input("Masukkan nilai faktorial yang dicari: "))
    print("Faktorial dari",t,"adalah", Faktorial(t))

main()
```

```
# Output
Masukkan nilai faktorial yang dicari: 5
Faktorial dari 5 adalah 120
```

2. Rekursi & Deret Fibonacci

Deret Fibonacci diambil dari nama seorang ahli matematika asal Italia, yaitu **Leonardo Fibonacci** (1170). Berikut adalah deret Fibonacci:

```
0,1,1,2,3,5,8,13,21,34,56,...
```

atau

```
if n=0          fib(n)=0
if n=1          fib(n)=1
if n>1          fib(n)=fib(n-1)+fib(n-2)
```

Contoh Rekursi & Deret Fibonacci :

```
# Fungsi rekursi & Deret Fibonacci

def Fibonacci(t):                   # Fungsi deret fibonacci
    if t==0:
        return 0
    elif t==1:
        return 1
    else:
        return Fibonacci(t-1)+Fibonacci(t-2)     # Rekursi fungsi deret fibonacci

def main():
    print("10 angka pertama deret Fibonacci:")
    for t in range(0,10):
        print(Fibonacci(t))

main()
```

```
# Output
10 angka pertama deret Fibonacci:
0
1
1
2
3
5
8
13
21
34
```

3. Rekursi & GCD (Greatest Common Divisor)

Greatest Common Divisor (GCD) atau dalam bahasa Indonesia dikenal dengan **Faktor Persekutuan Terbesar** (FPB) dari dua buah bilangan integer positif x dan y dapat ditentukan dengan syarat-syarat sebagai berikut:

```
if x%y = 0 → gcd(x,y) = y
else       → gcd(x,y) = gcd(y,x%y)
```

Contoh Rekursi & GCD (Greatest Common Divisor):

```
def GCD(t,q):
    if t % q==0:
        return q
    else:
        return GCD(q,t%q)

def main():
    pilihan="y"
    while(pilihan=='y' or pilihan=='Y'):
        t=int (input("Masukkan bilangan pertama: "))
        q=int (input("Masukkan bilangan kedua: "))
        print ("GCD dari ",t," dan ",q," adalah ",GCD(t,q))
        pilihan=input("Cari GCD lagi? (y/n): ")

main()            # Panggil fungsi utama
```

```
# Output
Masukkan bilangan pertama: 70
Masukkan bilangan kedua: 30
GCD dari  70  dan  30  adalah  10
Cari GCD lagi? (y/n):
```

Terima Kasih