

MODUL MATA KULIAH

BAHASA PEMROGRAMAN DASAR

PG168 – 3 SKS



**FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS BUDI LUHUR**

**JAKARTA
SEPTEMBER 2021**

TIM Penyusun

Agus Umar Hamdani, M.Kom
Tri Ika Jaya Kusumawati, M.Kom¹



MODUL PERKULIAHAN #9 PROCEDURE DAN FUNCTION

Capaian Pembelajaran	:	Mahasiswa Mampu: <ol style="list-style-type: none">1. Memahami konsep dasar prosedur (<i>procedure</i>) dan fungsi (<i>function</i>) dalam bahasa Python.2. Memahami karakteristik dan perbedaan prosedur dan fungsi.3. Memahami perbedaan User-Defined Function and Built-In Function4. Memahami fungsi-fungsi Tingkat Lanjut : Fungsi Map(), Fungsi Filter dan Fungsi Anonim (Lambda).
Sub Pokok Bahasan	:	<ol style="list-style-type: none">1. Procedure2. <i>Function</i>3. User-Defined Function vs Built-In Function.4. Pengenalan Fungsi-fungsi Tingkat Lanjut dalam Python.
Daftar Pustaka	:	<ol style="list-style-type: none">1. Zarman, Wendi dan Wicaksono, Mochamad Fajar. "<i>Implementasi Algoritma dalam bahasa Python</i>". Edisi Pertama. Bandung : Penerbit Informatika. 2020.2. Kurniawati, Arik. "Algoritma dan Pemrograman menggunakan Python". Edisi Pertama. Yogyakarta : Depublish. 2016.3. Ismah. "Pemrograman Komputer Dasar-dasar Python". Jakarta : Fakultas Ilmu Pendidikan Universitas Muhammadiyah Jakarta. 2017.4. Irfani, M. Haviz dan Dafid. "Modul Praktikum Dasar Pemrograman dengan bahasa Python". Palembang : Sekolah Tinggi Manajemen Informatika Global Informatika Multidata Palembang. 2016.5. Fikri, Rijalul. "Praktikum Algoritma dan Pemrograman Komputer". Surabaya : Program Studi Teknik Komputer dan Telematika Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember. 2010.6. Wiratmaja, I Gede Harjumawan, et.all. 2021. Program Menghitung Banyak Bata pada Ruangan Menggunakan Bahasa Python. TIERS Information Technology Journal. Vol. 2(1). Undiknas.7. Nuraini, Rini. 2017. Desk Check Table Pada Flowchart Operasi Perkalian Matriks. Jurnal Petir. Vol. 10(1). Sekolah Tinggi Teknik – PLN (STT-PLN).8. Romzi, Muhammad dan Kurniawan, Budi. 2020. Pembelajaran Pemrograman Python Dengan Pendekatan Logika Algoritma. JTIM : Jurnal Teknik Informatika Mahakarya. Vol. 03(2). Hal. 37-44.9. Programiz.com. Python Operators (https://www.programiz.com/python-programming/operators diakses pada 29 September 2021 pukul 21.32 WIB)

PRAKTIKUM 9

PROSEDUR DAN FUNGSI

9.1 Teori Singkat

Salah satu langkah yang baik dalam menyelesaikan persoalan yang kompleks adalah dengan membaginya ke dalam beberapa bagian program. Dalam pemrograman, pembagian program ke dalam bagian yang lebih kecil dilakukan dengan membaginya ke dalam beberapa subrutin. Pemrograman dengan pendekatan ini disebut dengan pemrograman modular.

Setiap bahasa pemrograman biasanya memiliki subrutin. Subrutin adalah suatu blok program (subprogram) yang terdiri dari beberapa perintah untuk menyelesaikan suatu masalah. Dalam bahasa Python dikenal dua buah subrutin, yaitu prosedur (*procedure*) dan fungsi (*function*). Adapun perbedaan prosedur dan fungsi adalah :

1. Fungsi selalu mengembalikan suatu nilai, ketika dipanggil. Sedangkan prosedur tidak mengembalikan nilai.
2. Eksekusi fungsi dilakukan tidak mandiri (bagian dari suatu perintah / statement). Sedangkan eksekusi prosedur dilakukan secara mandiri.
3. Fungsi tidak dapat memanggil prosedur. Sedangkan prosedur dapat memanggil fungsi.

9.2 Prosedur

Prosedur (*Procedure*) adalah blok / subprogram yang dapat dipanggil dalam menu utama (*main program*). terdapat dua jenis prosedur, yaitu : prosedur tanpa parameter dan prosedur dengan parameter. Secara umum sintak penulisan prosedur mirip dengan sintak penulisan suatu program yang terdiri dari tiga unsur utama, yaitu : nama, deklarasi dan aksi.

Sintak umum prosedur pada bahasa Python tanpa parameter adalah :

```
def Procedurename ( ) :  
    var (s)  
    statement (s)  
  
Procedurename( )
```

Keterangan :

1. *def* adalah kata kunci yang digunakan untuk membuat prosedur pada program.
2. *var (s)* adalah deklarasi variabel yang akan digunakan, bisa lebih dari satu variabel yang dibuat.
3. *Statement (s)* adalah deklarasi perintah atau aksi yang akan dijalankan kepada prosedur dipanggil atau dieksekusi.
4. Terdapat dua cara terkait pemanggilan prosedur, yaitu pemanggilan prosedur yang berada di dalam file yang sama dengan pemanggil dan pemanggilan prosedur yang berada di file yang berbeda. Berikut ini adalah caranya :
 - a. Untuk memanggil prosedur yang berada di dalam file yang sama, kita hanya perlu menuliskan nama prosedur yang ingin dipanggil disertai dengan simbol buka kurung tutup ().
 - b. Untuk memanggil prosedur yang berada di file yang berbeda, kita perlu melakukan import file yang berisi prosedur tersebut, baru kemudian memanggil prosedur yang ingin dieksekusi dengan cara menulis nama file yang diimport disertai dengan nama prosedur yang ingin dipanggil diikuti dengan tanda buka kurung tutup ().

9.3 Prosedur Dalam Satu File

Pembuatan prosedur dalam satu file sangat relative mudah, karena kita hanya perlu memanggil nama prosedur tersebut pada saat akan memulai proses eksekusi. Beberapa contoh kasus penerapan prosedur dalam satu file dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.1

Program pada gambar 9.1 merupakan contoh pembuatan prosedur dalam satu file yang samaa dengan nama "**prosedurKu.py**".

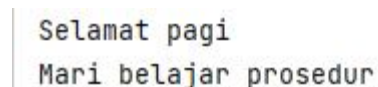


```
1  ▶
2  #deklarasi prosedur
3  def prosedurKu() :
4      A = "Selamat pagi"
5      print(A)
6      print("Mari belajar prosedur")
7
8  prosedurKu()
9
```

Gambar 9.1 Contoh Deklarasi Prosedur

Perhatikan gambar 9.1, terdapat deklarasi prosedur dengan nama "**prosedurKu**". Kemudian prosedur tersebut diisi dengan variabel **A** bertipe String dengan nilai "Selamat pagi". Kemudian terdapat perintah `print()` untuk mencetak variabel **A** dan mencetak kalimat "Mari belajar prosedur". Kemudian diluar blok prosedur tersebut terdapat perintah untuk memanggil nama prosedur, yaitu **prosedurKu()**.

Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.2.



```
Selamat pagi
Mari belajar prosedur
```

Gambar 9.2 Hasil Keluaran Program

9.4 Prosedur Pada File Berbeda dan Direktori Sama

Pada bagian ini, kita akan mempelajari bagaimana membuat prosedur dan memanggil prosedur yang disimpan di file berbeda. Beberapa contoh kasus penerapan prosedur dalam satu file dapat dilihat pada contoh-contoh studi kasus dibawah ini.

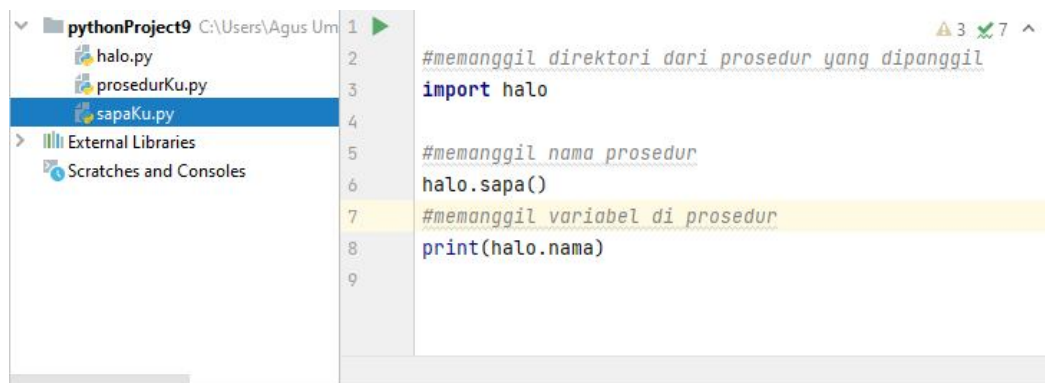
Studi Kasus 9.3

Program pada gambar 9.3 merupakan contoh pembuatan prosedur dalam file yang berbeda, tetapi masih dalam direktori yang sama.



Gambar 9.3 Contoh Deklarasi Prosedur

Perhatikan gambar 9.3, terdapat deklarasi prosedur dengan nama **"halo.py"**. Kemudian prosedur tersebut diisi dengan perintah `print()` untuk mencetak kalimat "Hallo, Selamat pagi, Salam Budi Luhur". Kemudian diluar blok prosedur tersebut terdapat variabel `nama` dengan tipe String yang berisi nilai "Abdullah". Prosedur tersebut akan dipanggil ke dalam file baru yang bernama **"sapaKu.py"**, selengkapnya dapat dilihat pada gambar 9.4.



Gambar 9.4 Memanggil Prosedur dari File Berbeda

Perhatikan gambat 9.4, terdapat deklarasi file dengan nama **"sapaKu.py"**, kemudian terdapat perintah **"import halo"**, perintah ini digunakan untuk memanggil file **"halo.py"**. Kemudian prosedur `sapa` yang ada di file **"halo.py"** dipanggil dengan perintah **"halo.sapa()"**. Setelah nama prosedur dipanggil, maka variabel-variabel yang ada di dalam prosedur tersebut dapat ikut dipanggil juga.

Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.5.

```
Hallo, Selamat pagi, Salam Budi Luhur  
Abdullah
```

Gambar 9.5 Hasil Keluaran Program

9.5 Variabel Lokal dan Variabel Global

Selama ini kita hanya membahas variabel, tanpa mengetahui jenis dari variabel tersebut. Terdapat dua jenis variabel berdasarkan cakupan / ruang lingkup (scope) setiap variabelnya, yaitu :

- a. Variabel Global (*Global Variable*) adalah variabel yang dapat digunakan di seluruh bagian program.
- b. Variabel Lokal (*Local Variable*) adalah variabel yang hanya dapat digunakan pada subprogram tertentu saja.

Perbedaan antara Variabel Global dan Variabel Lokal terletak pada tempat variabel tersebut dideklarasikan. Jika suatu variabel dideklarasikan di dalam prosedur atau di dalam fungsi, maka variabel tersebut disebut Variabel Lokal, artinya variabel tersebut hanya dapat digunakan di dalam prosedur atau fungsi yang bersangkutan. Sebaliknya, jika variabel dideklarasikan di luar prosedur dengan sendirinya disebut Variabel Global, artinya dapat digunakan di seluruh badan program, termasuk juga di dalam prosedur itu sendiri dengan catatan deklarasi variabel itu ditulis sebelum membuat deklarasi prosedur. Kesalahan dalam memanfaatkan Variabel Lokal dan Variabel Global akan mengakibatkan kesalahan program (error).

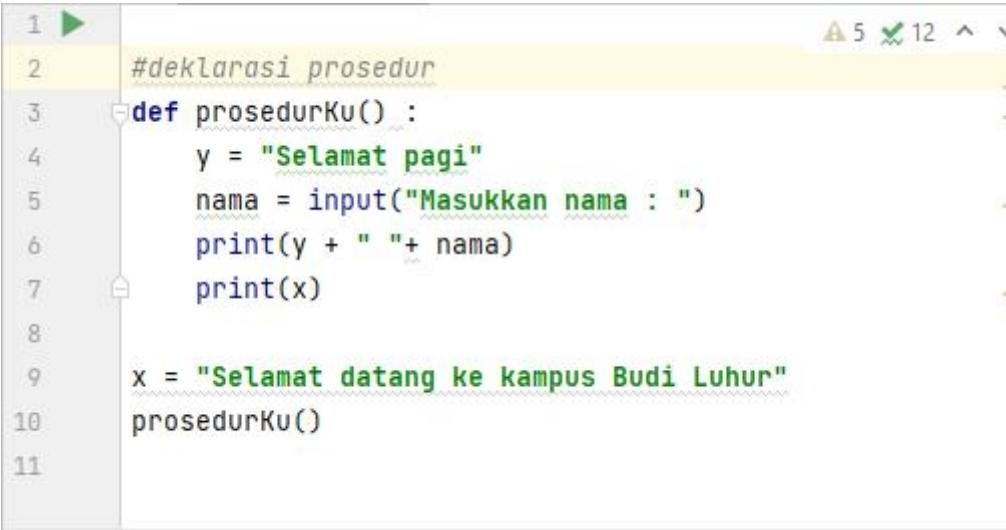
Sintak umum prosedur pada bahasa Python tanpa parameter adalah :

```
def Procedurename ( ) :  
    local_var  
    statement (s)  
  
global_var  
Procedurename( )
```


Variabel "local_var" berada di dalam prosedur dan hanya dapat digunakan di dalam prosedur saja. Jika variabel tersebut diakses ataupun diproses di luar prosedur, maka akan menyebabkan eror pada program, karena dianggap ilegal. Variabel "global_var" dapat diakses dimana saja pada program, baik di luar prosedur atau fungsi maupun di dalam prosedur atau fungsi. Beberapa contoh kasus penerapan Variabel Global dan Variabel Lokal dapat dilihat pada contoh-contoh studi kasus dibawah ini.

Studi Kasus 9.4

Program pada gambar 9.6 merupakan contoh pembuatan variabel global dan variabel local di dalam prosedur "**prosedurKu.py**".



```
1  #deklarasi prosedur
2
3  def prosedurKu() :
4      y = "Selamat pagi"
5      nama = input("Masukkan nama : ")
6      print(y + " " + nama)
7      print(x)
8
9  x = "Selamat datang ke kampus Budi Luhur"
10 prosedurKu()
11
```

Gambar 9.6 Contoh Deklarasi Variabel Global dan Variabel Lokal

Perhatikan gambar 9.6, terdapat deklarasi prosedur dengan nama "**prosedurKu**". Kemudian prosedur tersebut diisi dengan variabel **y** bertipe String dengan nilai "Selamat pagi" dan terdapat variabel nama dengan perintah `user_input()` bertipe String yang menerima nilai inputan dari user. Variabel **y** dan **nama** disebut dengan variabel local (*Local Variabel*). Kemudian terdapat perintah `print()` untuk mencetak variabel **y** dan **nama**. Kemudian terdapat deklarasi variabel **x** yang berada diluar blok prosedur dan bertipe String dengan nilai "Selamat datang ke kampus Budi Luhur". Variabel **x** disebut dengan variabel global (*Global Variable*). Kemudian diluar blok prosedur tersebut terdapat perintah untuk memanggil nama prosedur, yaitu **prosedurKu()**.

Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.6.

```
Masukkan nama : Abdullah
Selamat pagi Abdullah
Selamat datang ke kampus Budi Luhur
```

Gambar 9.6 Hasil Keluaran Program

9.6 Parameter Nilai dan Parameter Variabel

Telah disebutkan sebelumnya bahwa jika suatu prosedur membutuhkan informasi agar dapat bekerja / dapat dieksekusi, maka informasi tersebut diperoleh dengan melalui parameter yang dituliskan pada deklarasi prosedurnya. Parameter ini disebut dengan **parameter formal (*formal parameter*)**. Nilai parameter formal ini dikirim dari **parameter nyata (*real parameter*)**, yaitu parameter yang terletak di dalam program utama. Terdapat dua cara pengiriman parameter, yaitu :

- Pengiriman secara nilai : menggunakan nilai untuk memanggil sebuah prosedur atau fungsi.
- Pengiriman secara acuan (variabel) : menggunakan variabel tertentu untuk memanggil sebuah prosedur atau fungsi.

Bentuk umum deklarasi prosedur dengan parameter nilai adalah :

```
def Procedurename ( formal_parameter-1, formal_parameter-2, ...) :  
    local_var  
    statement (s)  
  
global_var  
Procedurename( real_parameter-1, real_parameter-2, ...)
```

Struktur ini mirip dengan struktur prosedur tanpa parameter, perbedaannya adalah terdapat parameter yang bisa berupa *immediate* data atau berupa variabel. Parameter-parameter ini yang nantinya akan diproses di dalam prosedur.

Beberapa contoh kasus penerapan Parameter Nilai dan Parameter Variabel dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.7

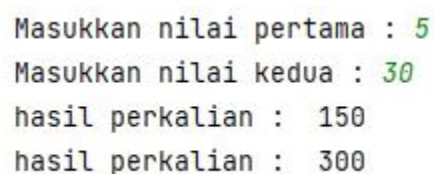
Program pada gambar 9.7 merupakan contoh penggunaan parameter nyata berupa nilai pada prosedur "**perkalian1.py**".



```
1  ▶
2  #deklarasi prosedur
3  def perkalian(angka1, angka2) :
4      hasil = angka1 * angka2
5      print("hasil perkalian : ", hasil)
6
7  perkalian(100, 300)
8
9
```

Gambar 9.7 Contoh Deklarasi Prosedur dengan Parameter

Perhatikan gambar 9.7, terdapat deklarasi prosedur dengan nama "**perkalian**". Prosedur tersebut memiliki dua parameter variabel dengan nama **angka1** dan **angka2**. Kemudian prosedur tersebut diisi dengan variabel **hasil** yang merupakan perkalian antara variabel **angka1** dan **angka2**. Kemudian terdapat perintah **print()** untuk mencetak nilai dari variabel **hasil**. Kemudian diluar blok prosedur tersebut terdapat perintah untuk memanggil prosedur **perkalian** disertai pemberian **nilai pada angka1 = 100** dan **nilai pada angka2 = 300**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.10.



```
Masukkan nilai pertama : 5
Masukkan nilai kedua : 30
hasil perkalian : 150
hasil perkalian : 300
```

Gambar 9.10 Hasil Keluaran Program

Studi Kasus 9.11

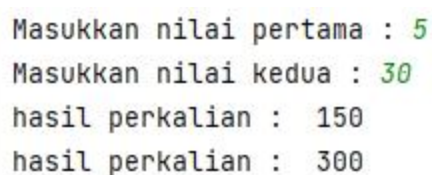
Program pada gambar 9.11 merupakan contoh penggunaan parameter nyata berupa variabel pada prosedur "**perkalian2.py**".



```
1  ▶  
2  #deklarasi prosedur  
3  def perkalian2(angka1, angka2) :  
4      hasil = angka1 * angka2  
5      print("hasil perkalian : ", hasil)  
6  
7  a = int(input("Masukkan nilai pertama : "))  
8  b = int(input("Masukkan nilai kedua : "))  
9  perkalian2(a, b)  
10  
11  c = 20  
12  d = 15  
13  perkalian2(c, d)  
14
```

Gambar 9.11 Contoh Deklarasi Prosedur dengan Parameter

Perhatikan gambar 9.11, terdapat deklarasi prosedur dengan nama "**perkalian2**". Prosedur tersebut memiliki dua parameter dengan nama **angka1** dan **angka2**. Kemudian prosedur tersebut diisi dengan variabel **hasil** yang merupakan perkalian antara variabel **angka1** dan **angka2**. Kemudian terdapat perintah **print()** untuk mencetak nilai dari variabel **hasil**. Kemudian diluar blok prosedur tersebut terdapat dua variabel, yaitu **a**, **b**, **c** dan **d**. variabel **a** dan **b** bertipe **Integer** yang menggunakan perintah **user_input()** untuk menangkap nilai dari keyboard. Sedangkan variabel **c** dan **d** bertipe **Integer** dengan nilai tetap, yaitu **c = 20** dan **d = 15**. Variabel-variabel global tersebut memanggil prosedur **perkalian2** dan kemudian mencetak hasilnya. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.12.




```
Masukkan nilai pertama : 5  
Masukkan nilai kedua : 30  
hasil perkalian : 150  
hasil perkalian : 300
```

Gambar 9.12 Hasil Keluaran Program

Studi Kasus 9.13

Program pada gambar 9.13 merupakan contoh penggunaan operator perulangan For dan parameter nyata berupa variabel pada prosedur "**perkalian3.py**".



```
1  ▶
2  #deklarasi prosedur
3  def perkalian3(angka1) :
4      hasil = angka1 * 5
5      print("hasil perkalian : ", hasil)
6
7  jumlah = int(input("Masukan jumlah perulangan : "))
8  for i in range(jumlah) :
9      x = int(input("Masukan bilangan = "))
10     perkalian3(x)
11
12
```

Gambar 9.13 Contoh Deklarasi Prosedur dengan Operator Perulangan

Perhatikan gambar 9.13, terdapat deklarasi prosedur dengan nama "**perkalian3**". Prosedur tersebut memiliki sebuah parameter dengan nama **angka1**. Kemudian prosedur tersebut diisi dengan variabel **hasil** yang merupakan perkalian antara variabel **angka1** dengan angka **5**. Kemudian terdapat perintah **print()** untuk mencetak nilai dari variabel **hasil**. Diluar blok prosedur tersebut terdapat variabel **jumlah** bertipe **Integer** yang menggunakan perintah **user_input()** untuk menangkap nilai dari keyboard. Kemudian terdapat struktur perulangan **for** dengan batas akhir berulang adalah nilai yang terdapat pada variabel **jumlah**. Di dalam perulangan **for** terdapat variabel **x** bertipe **Integer** dan bersifat variabel global yang menerima inputan dari keyboard. Kemudian terdapat perintah untuk memanggil prosedur **perkalian3** dengan parameter berupa variabel **x**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.14.

Studi Kasus 9.15

Program pada gambar 9.15 merupakan contoh penggunaan variabel global dengan beberapa prosedur berbeda pada file "**ProsedurGabung.py**".



```

1  ▶
2  print("Program menghitung X Pangkat N")
3  x=0
4  n=0
5  pangkat=1
6
7  def masukan() :
8      global x, n
9      x = int(input("Masukkan nilai x : "))
10     n = int(input("Masukkan nilai n : "))
11
12     def hitung() :
13         global x, n, pangkat
14         for i in range(n) :
15             pangkat = pangkat * x
16
17     def tampil() :
18         print(x, " pangkat ", n, "=", pangkat)
19
20     masukan()
21     hitung()
22     tampil()

```

Gambar 9.15 Contoh Penggunaan Variabel Global pada Prosedur Berbeda

Perhatikan gambar 9.15, terdapat deklarasi prosedur dengan nama **"ProsedurGabung"**. Di awal program, variabel x , n dan $pangkat$ adalah **Variabel Global (*Global Variable*)**. Ingat, pada saat kita melakukan modifikasi nilai pada variabel yang memiliki nama sama, maka variabel tersebut akan dianggap sebagai **Variabel Lokal (*Local Variable*)**. Agar variabel x dan n dianggap sebagai Variabel Global, maka kita perlu menggunakan **keyword Global** pada variabel-variabel tersebut.

Pada saat program dijalankan, maka program akan memanggil prosedur **masukan()**. Pada saat pengguna menginput nilai x dan n yang ada di prosedur masukan, maka aktivitas tersebut dianggap legal dilakukan dan nilai variabel global x dan n yang semula bernilai 0 akan diubah sesuai dengan inputan yang diberikan oleh pengguna. Nilai masukan x dan n bersifat Global, meskipun proses perubahannya terjadi di dalam prosedur sehingga nilai x dan n untuk seluruh program sekarang adalah nilai yang sesuai dengan masukan dari pengguna.

Selanjutnya, program akan memanggil prosedur `hitung()` untuk menghitung nilai x pangkat n . Semua variabel global yang akan dimodifikasi pada prosedur ini juga perlu diberikan keyword `Global`. Pada prosedur ini dilakukan operasi perhitungan x pada n yang hasilnya disimpan ke variabel global pangkat.

Bagian akhir dari program ini adalah pemanggilan prosedur `tampil()`. Saat prosedur `tampil` dipanggil, program akan mengeksekusi perintah pencetakan nilai hasil perpangkatan yang disimpan di variabel pangkat. Pada posisi ini, variabel pangkat tidak dideklarasikan sebagai variabel global, karena tidak ada modifikasi nilai. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.16.

```
Program menghitung X Pangkat N
Masukkan nilai x : 3
Masukkan nilai n : 5
3 pangkat 5 = 243
```

Gambar 9.16 Hasil Keluaran Program

9.7 Fungsi (Function)

Fungsi pada dasarnya sama seperti prosedur. Perbedaannya adalah fungsi selalu mengembalikan nilai pada saat dipanggil, sedangkan prosedur tidak demikian. Dengan menggunakan fungsi, kita dapat menulis blok kode program yang dapat digunakan Kembali dengan cara yang berbeda dan dalam program yang berbeda. Pada materi ini akan dibahas pembuatan fungsi sesuai kebutuhan pengguna atau disebut dengan ***User-Defined Functions***.

Terdapat beberapa Langkah untuk mendefinisikan fungsi dalam bahasa pemrograman Python, yaitu :

- Fungsi diawali dengan kata kunci `def` diikuti dengan **nama fungsi** dan simbol buka tutup kurung `()` dan diberi tanda `:`.
- Parameter input atau argument ditempatkan di dalam simbol buka tutup kurung `()`. Parameter ini mempunyai positional behavior sehingga kita perlu menginformasikan urutan yang sama, dimana parameter tersebut didefinisikan atau tanpa urutan yang sama, tetapi pada ekspresi pemanggil

menggunakan nama parameter yang sama dengan nama parameter yang ada pada fungsi.

- c. Semua pernyataan (*statements*) dari fungsi harus diberi indentasi.
- d. Pernyataan paling awal dari suatu fungsi bisa berupa pernyataan opsional yang disebut dengan ***Docstring (Documentation String)***. Pernyataan tersebut berguna untuk mendokumentasikan fungsi agar mudah dipahami dan digunakan.
- e. Pernyataan `return` digunakan untuk keluar dari fungsi dan akan mengembalikan nilai ke ekspresi pemanggil fungsi. Pernyataan ini bersifat opsional.

Bentuk umum deklarasi fungsi dengan parameter nilai adalah :

```
def Functionname ( formal_parameter-1, formal_parameter-2, ...) :  
    #Docstring  
    statement (s)  
    return value  
  
Functionname( real_parameter-1, real_parameter-2, ...)
```

Berdasarkan struktur fungsi diatas, maka cara untuk memanggil fungsi adalah dengan menuliskan nama fungsi yang akan dipanggil diikuti dengan parameter yang diperlukan atau tanpa diikuti parameter. Beberapa contoh kasus penerapan fungsi dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.17

Program pada gambar 9.17 merupakan contoh pemanggilan fungsi tanpa parameter pada fungsi "**fungsiTP.py**".


```

1  ▶
2  #definisi fungsi
3  def angka() :
4      x = 20 * 5
5      return x
6
7  nilai = angka()
8  print(nilai)
9

```

Gambar 9.17 Contoh Deklarasi Fungsi Tanpa Parameter

Perhatikan gambar 9.17, terdapat deklarasi fungsi dengan nama “**angka**”. fungsi tersebut tidak memiliki parameter sehingga pemanggilan fungsi angka hanya dengan menggunakan nama fungsinya saja. Terdapat deklarasi variabel **x** bertipe Integer untuk menyimpan hasil perkalian antara angka **20** dan **5**. Terdapat perintah **return x** untuk mengembalikan hasil dari variabel **x**. Kemudian diluar blok fungsi tersebut terdapat variabel **nilai** bertipe **Integer** yang akan digunakan untuk menyimpan nilai dari fungsi angka dan kemudian mencetak hasil yang terdapat pada variabel **nilai**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.18.

```

100

```

Gambar 9.18 Hasil Keluaran Program

Studi Kasus 9.19

Program pada gambar 9.19 merupakan contoh pemanggilan fungsi yang disertai dengan parameter pada fungsi “**jumlah.py**”.

```

1  ▶
2  #definisi fungsi
3  def jumlah(a, b) :
4      c = a + b
5      return c
6
7  angka1 = int(input("Masukkan angka pertama : "))
8  angka2 = int(input("Masukkan angka kedua : "))
9  hasil = jumlah(angka1, angka2)
10 print(angka1, " + ", angka2, "=", hasil)
11

```

Gambar 9.19 Contoh Deklarasi Fungsi dengan Parameter

Perhatikan gambar 9.19, terdapat deklarasi fungsi dengan nama "**jumlah**". Fungsi tersebut memiliki dua parameter, yaitu variabel **a** dan **b**. Terdapat deklarasi variabel **c** bertipe **Integer** untuk menyimpan hasil penjumlahan antara variabel **a** dan variabel **b**. Terdapat perintah **return c** untuk mengembalikan dari variabel **c**. Kemudian diluar blok prosedur tersebut terdapat variabel **angka1** dan **angka2** bertipe **Integer** dan berisi perintah `user_input()` yang akan merekam inputan dari keyboard. Kemudian terdapat deklarasi variabel **hasil** yang akan memanggil fungsi **jumlah** dengan parameter berupa nilai dari variabel **angka1** dan **angka2**. Kemudian terdapat perintah **print()** untuk mencetak hasil yang terdapat pada variabel nilai. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.20

```

Masukkan angka pertama : 20
Masukkan angka kedua : 30
20 + 30 = 50

```

Gambar 9.20 Hasil Keluaran Program

9.8 Parameter Default

Dalam bahasa pemrograman Python dikenal istilah **Parameter Default**. **Parameter default** merupakan sebuah nilai yang diasumsikan sebagai nilai default untuk fungsi, jika parameter yang diperlukan oleh fungsi tidak **parameter** disertakan oleh pengguna pada saat memanggil fungsi. Hal tersebut dapat digunakan untuk menghindari kesalahan (error) pada program Ketika kita secara tidak sengaja lupa memasukkan parameter-parameter pada pemanggil fungsi. Penggunaan Parameter Default ini sama dengan pemberian nilai pada sebuah variabel. Beberapa contoh kasus penerapan fungsi dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.21

Program pada gambar 9.21 merupakan contoh menggunakan fungsi dengan parameter default pada fungsi "**fungsiKu.py**".



```
1  ▶
2  #definisi fungsi
3  def fungsiKu(nama = "xyz", nomor = 0) :
4      print("Haloo ", nama + ", Nomor urut : ", nomor)
5
6      fungsiKu("Bambang Jeger", 3)
7      fungsiKu("")
8      fungsiKu("Joe Taslim", 10)
9
```

Gambar 9.21 Contoh Deklarasi Fungsi dengan Parameter Default

Perhatikan gambar 9.21, terdapat deklarasi fungsi dengan nama "**fungsiKu**". Fungsi tersebut memiliki dua parameter, yaitu variabel **nama** dan **nomor**. Terdapat perintah **print()** untuk mencetak kalimat "**Haloo**", nilai dari **variabel nama** dan nilai dari **variabel nomor**. Kemudian diluar blok fungsi tersebut terdapat tiga perintah yang memanggil fungsi **fungsiKu** dengan dua parameter, yaitu **nama** dan **nomor**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.22

```
Haloo Bambang Jeger, Nomor urut : 3
Haloo , Nomor urut : 0
Haloo Joe Taslim, Nomor urut : 10
```

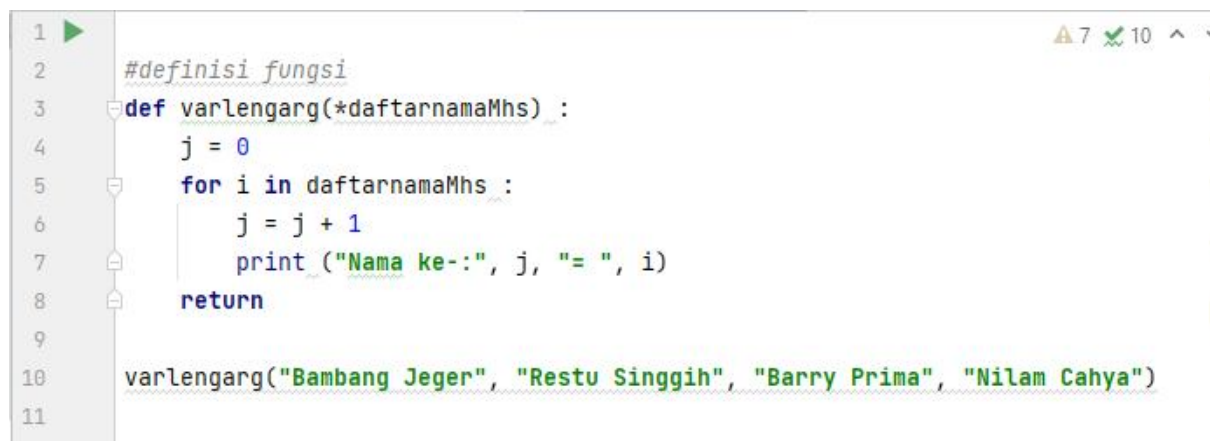
Gambar 9.22 Hasil Keluaran Program

9.9 Variable-Length Argument

Dalam bahasa pemrograman Python dikenal istilah **Variable-Length Argument**. **Variable-Length** digunakan untuk mengakses list / daftar dari parameter pemanggil. Penulisan parameter formal pada fungsi harus diawali dengan tanda asterisk (*) yang akan menunjukkan pada bahasa Python bahwa argument atau parameter adalah sebuah List. Parameter ini akan berlaku seperti tipe List Standar yang biasa digunakan, Ketika berada di dalam sebuah fungsi. Beberapa contoh kasus penerapan fungsi dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.23

Program pada gambar 9.23 merupakan contoh penggunaan *variable-length argument* pada fungsi "varlengarg.py".



```
1  #definisi fungsi
2
3  def varlengarg(*daftarnamaMhs) :
4      j = 0
5      for i in daftarnamaMhs :
6          j = j + 1
7          print ("Nama ke-:", j, "=", i)
8      return
9
10 varlengarg("Bambang Jeger", "Restu Singgih", "Barry Prima", "Nilam Cahya")
11
```

Gambar 9.23 Contoh Penggunaan Variable-Length Argument

Perhatikan gambar 9.23, terdapat deklarasi fungsi dengan nama "varlengarg". Fungsi tersebut memiliki sebuah parameter, berupa variabel ***daftarnamaMhs** yang merupakan **Variable-Length Argument** sehingga nanti fungsi mengenali

bahwa parameter nyata yang dilepaskan ke fungsi adalah sebuah List. Hal ini dibuktikan dengan perulangan yang dilakukan, dimana range dari perulangan menggunakan variabel `daftarnamaMhs`. Kemudian diluar blok fungsi tersebut terdapat perintah untuk memanggil fungsi **varlengarg** dengan dengan nilai = ["Bambang Jeger", "Restu Singgih", "Barry Prima", "Nilam Cahya"] . Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.24.

```
Haloo Bambang Jeger, Nomor urut : 3
Haloo , Nomor urut : 0
Haloo Joe Taslim, Nomor urut : 10
```

Gambar 9.24 Hasil Keluaran Program

9.10 Fungsi Standar

Dalam bab sebelumnya kita membahas pembuatan fungsi yang didefinisikan sesuai kebutuhan pengguna atau disebut **User-Defined Functions**. Bahasa Python juga menyediakan banyak fungsi yang siap pakai (**Built-In**) yang dapat dikategorikan ke dalam beberapa bagian sesuai dengan fungsionalitasnya, yaitu *input / output*, *encoding character*, *math* (matematika), pengurutan dan iterasi. Tabel 1 berikut ini menunjukkan beberapa contoh umum dari **fungsi Built-In** yang ada di dalam bahasa pemrograman Python.

Tabel 1. Built-In Functions pada Bahasa Python

No.	Nama <i>Built-In Functions</i>	Keterangan
1.	<code>abs(x)</code>	Mengembalikan nilai absolut dari sebuah bilangan.
2.	<code>all(iterable)</code>	Mengembalikan nilai Boolean True, jika semua elemen bernilai True.
3.	<code>Any(iterable)</code>	Mengembalikan nilai Boolean True, jika minimal terdapat satu elemen bernilai True.

4.	<code>Chr(integer)</code>	Mengembalikan sebuah string satu karakter sesuai dengan titik kode Unicode yang diberikan.
5.	<code>Divmod(x,y)</code>	Mengembalikan nilai $(x/y, x\%y)$ sebagai tuple, jika x dan y adalah bilangan bulat.
6.	<code>Hash(obj)</code>	Mengembalikan nilai hash dalam bentuk integer untuk obyek.
7.	<code>Id(obj)</code>	Mengembalikan nilai unik dalam bentuk Integer yang berfungsi sebagai identitas untuk obyek.
8.	<code>Input(prompt)</code>	Mengembalikan nilai string dari input standar.
9.	<code>Ininstance(obj, cls)</code>	Menentukan apakah obj merupakan turunan dari kelas (subclass).
10.	<code>Iter(iterable)</code>	Mengembalikan obyek iterator baru untuk parameter.
11.	<code>Len(iterable)</code>	Mengembalikan jumlah elemen dari iterasi yang diberikan.
12.	<code>Map(f, iter1, iter2, ...)</code>	Mengembalikan iterator yang menghasilkan hasil pemanggilan fungsi $f(e1, e2)$ untuk masing-masing e1 elemen dari iter1, e2 elemen dari iter2, dst).
13.	<code>Max(iterable)</code>	Mengembalikan nilai terbesar dari iterasi yang diberikan.
14.	<code>Max(a, b, c, ...)</code>	Mengembalikan nilai terbesar dari argumen-argumen yang diberikan.
15.	<code>Min(iterable)</code>	Mengembalikan nilai terkecil dari iterasi yang diberikan.
16.	<code>Min(a, b, c, ...)</code>	Mengembalikan nilai terkecil dari argumen-argumen yang diberikan.

17.	Next(iterator)	Mengembalikan elemen selanjutnya yang dilaporkan oleh iterator.
18.	Open(filename, mode)	Membuka file sesuai dengan nama file yang diberikan dan jenis akses mode yang diberikan.
19.	Ord(char)	Mengembalikan kode Unicode sesuai dengan karakter yang diberikan.
20.	Pow(x, y)	Mengembalikan nilai x pangkat y , dengan syarat x dan y bertipe Integer.
21.	Pow(x, y, z)	Mengembalikan nilai hasil $x^y \bmod z$ dalam bentuk Integer.
22.	Print(obj1, obj1, ...)	Mencetak setiap argument disertai baris baru.
23.	Range(stop)	Membangun sebuah iterasi yang dimulai dari 0, 1, ..., stop-1.
24.	Range(start, stop)	Membangun sebuah iterasi yang dimulai dari start, start + 1, ..., stop-1.
25.	Range(start, stop, step)	Membangun sebuah iterasi yang dimulai dari start, start + step, start + 2 * step, ..., stop-1.
26.	Reserved(sequence)	Mengembalikan iterasi dari sequence secara terbalik.
27.	Round(x)	Mengembalikan nilai integer terdekat.
28.	Round(x, k)	Mengembalikan nilai hasil pembulatan ke 10^k .
29.	Sorted(iterable)	Mengembalikan nilai sebuah list yang didalamnya mengandung elemen yang dapat diurutkan.
30.	Sum(iterable)	Mengembalikan hasil total penjumlahan dari elemen-elemen yang dapat diiterasi (elemen ini harus dalam bentuk numerik).

9.11 Fungsi Rekursif

Bahasa pemrograman Python juga menerima **Fungsi Rekursif** (*Recursive Function*). **Fungsi Rekursif** adalah fungsi yang dapat memanggil dirinya sendiri secara berulang-ulang hingga suatu kondisi yang di definisikan terpenuhi atau bernilai benar (*True*). Dengan adanya Fungsi Rekursif, maka kita dapat mengulang data untuk mencapai hasil sehingga menjadi model pemrograman yang sangat efisien dan elegan secara matematis. tetapi programmer harus berhati-hati, karena dapat dengan mudah tergelincir ke dalam penulisan fungsi yang tidak pernah berhenti atau fungsi yang menggunakan jumlah memori atau daya prosesor yang berlebihan. Fungsi Rekursif juga bisa disebut **perulangan rekursif** (karena fungsinya memang sama seperti looping) perbedaannya adalah jika pada perulangan iteratif menggunakan instruksi perulangan seperti `for` dan `while`, sementara pada fungsi rekursif perulangan dilakukan pada fungsi itu sendiri.

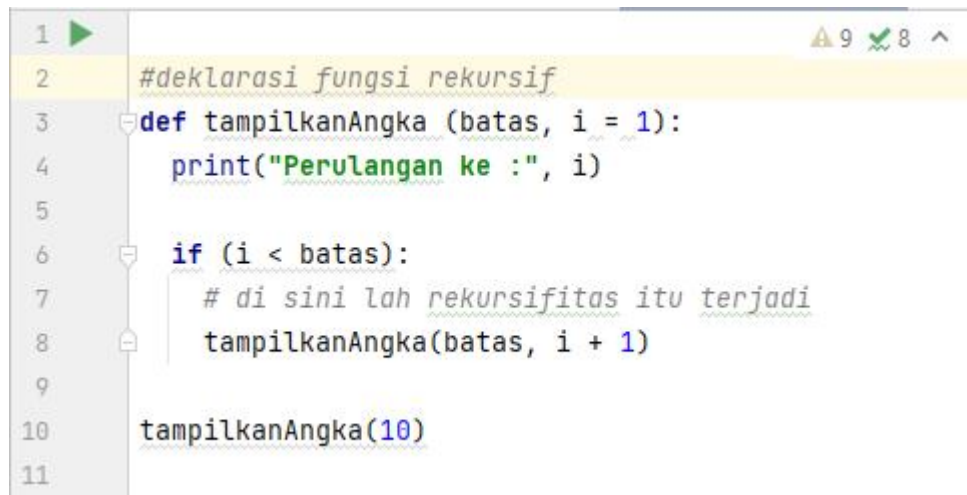
Berikut ini merupakan tiga contoh penerapan dari fungsi rekursif yaitu:

- a. Faktorial
- b. Fungsi Pangkat
- c. Fibonacci

Beberapa contoh kasus penerapan fungsi dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.25

Program pada gambar 9.25 merupakan contoh penggunaan *variable-length argument* pada file "**fungsiRekursif-1.py**".



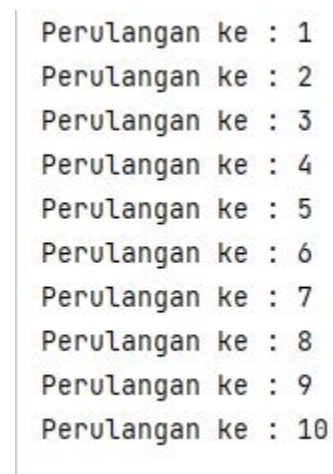
```

1  ▶
2  #deklarasi fungsi rekursif
3  def tampilkanAngka (batas, i = 1):
4      print("Perulangan ke :", i)
5
6      if (i < batas):
7          # di sini lah rekursifitas itu terjadi
8          tampilkanAngka(batas, i + 1)
9
10 tampilkanAngka(10)
11

```

Gambar 9.25 Contoh Penggunaan Fungsi Rekursif

Perhatikan gambar 9.25, terdapat deklarasi fungsi dengan nama **"tampilkanAngka"**. Fungsi tersebut memiliki dua parameter, yaitu variabel **batas** dan variabel **i**. Variabel **i** diberi nilai default 1. Terdapat kondisi percabangan IF untuk memeriksa nilai **i**, apakah kurang dari nilai **batas** atau tidak. Jika nilai **i** kurang dari nilai **batas**, maka fungsi **tampilkanAngka()** akan dipanggil kembali. Kemudian di luar blok fungsi tersebut terdapat perintah untuk memanggil fungsi **tampilkanAngka** disertai dengan parameter berupa nilai angka 10. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.26.



```

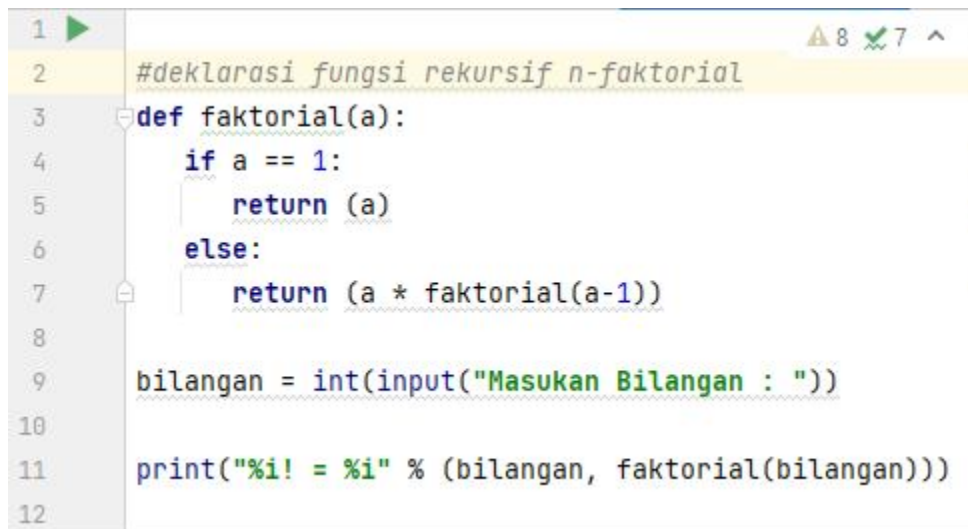
Perulangan ke : 1
Perulangan ke : 2
Perulangan ke : 3
Perulangan ke : 4
Perulangan ke : 5
Perulangan ke : 6
Perulangan ke : 7
Perulangan ke : 8
Perulangan ke : 9
Perulangan ke : 10

```

Gambar 9.26 Hasil Keluaran Program

Studi Kasus 9.26


Program pada gambar 9.26 merupakan contoh penggunaan fungsi rekursif pada file "**Faktorial.py**".



```
1  ▶ #deklarasi fungsi rekursif n-faktorial
2
3  def faktorial(a):
4      if a == 1:
5          return (a)
6      else:
7          return (a * faktorial(a-1))
8
9  bilangan = int(input("Masukan Bilangan : "))
10
11  print("%i! = %i" % (bilangan, faktorial(bilangan)))
12
```

Gambar 9.26 Contoh Penggunaan Fungsi Rekursif - Faktorial

Perhatikan gambar 9.26, terdapat deklarasi fungsi dengan nama "**factorial**". Fungsi tersebut memiliki sebuah parameter, yaitu variabel **a**. Terdapat kondisi percabangan **IF** untuk memeriksa **nilai a**, apakah sama dengan **1** atau tidak. Jika **nilai a sama dengan 1**, maka program akan **mengembalikan nilai a**. jika **nilai a tidak sama dengan 1**, maka program akan **mengembalikan hasil dari $a * \text{factorial}(a-1)$** . Di blok luar fungsi tersebut terdapat deklarasi variabel **bilangan** yang bertipe **Integer** yang akan menerima inputan nilai dari keyboard. Kemudian terdapat perintah untuk mencetak dan memanggil fungsi **factorial()** disertai dengan parameter berupa nilai dari variabel **bilangan**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.27.



```
Masukan Bilangan : 5
5! = 120
```

Gambar 9.27 Hasil Keluaran Program

Studi Kasus 9.28

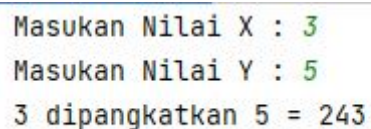
Program pada gambar 9.28 merupakan contoh penggunaan fungsi rekursif pada file "Pangkat.py".



```
1  ▶
2  #deklarasi fungsi rekursi pangkat
3  def pangkat(x,y):
4      if y == 0:
5          return 1
6      else:
7          return x * pangkat(x,y-1)
8
9  x = int(input("Masukan Nilai X : "))
10 y = int(input("Masukan Nilai Y : "))
11
12 print("%d dipangkatkan %d = %d" % (x,y,pangkat(x,y)))
13
```

Gambar 9.27 Contoh Penggunaan Fungsi Rekursif - Pangkat

Perhatikan gambar 9.27, terdapat deklarasi fungsi dengan nama "**pangkat**". Fungsi tersebut memiliki dua parameter, yaitu variabel **x** dan **y**. Terdapat kondisi percabangan **IF** untuk memeriksa **nilai y**, apakah sama dengan **0** atau tidak. Jika **nilai a sama dengan 0**, maka program akan **mengembalikan nilai 1**. jika **nilai a tidak sama dengan 0**, maka program akan **mengembalikan hasil dari $x * \text{pangkat}(x, y-1)$** . Di luar blok fungsi tersebut terdapat deklarasi variabel **x** dan **y** yang bertipe **Integer** yang akan menerima inputan nilai dari keyboard. Kemudian terdapat perintah untuk mencetak dan memanggil fungsi **pangkat()** disertai dengan parameter berupa variabel **x** dan **y**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.28.

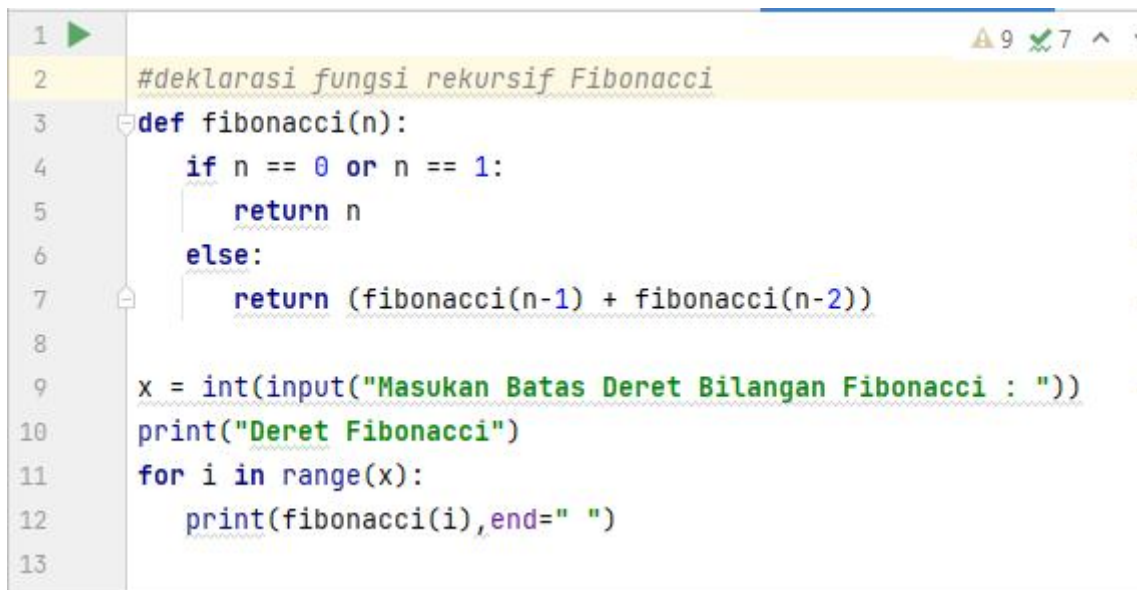


```
Masukan Nilai X : 3
Masukan Nilai Y : 5
3 dipangkatkan 5 = 243
```

Gambar 9.28 Hasil Keluaran Program

Studi Kasus 9.29

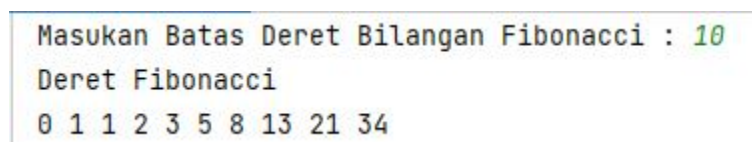
Program pada gambar 9.29 merupakan contoh penggunaan fungsi rekursif pada file "**Fibonacci.py**".



```
1  ▶
2  #deklarasi fungsi rekursif Fibonacci
3  def fibonacci(n):
4      if n == 0 or n == 1:
5          return n
6      else:
7          return (fibonacci(n-1) + fibonacci(n-2))
8
9  x = int(input("Masukan Batas Deret Bilangan Fibonacci : "))
10 print("Deret Fibonacci")
11 for i in range(x):
12     print(fibonacci(i),end=" ")
13
```

Gambar 9.29 Contoh Penggunaan Fungsi Rekursif - Fibonacci

Perhatikan gambar 9.29, terdapat deklarasi fungsi dengan nama "**fibonacci**". Fungsi tersebut memiliki sebuah parameter, yaitu variabel **n**. Terdapat kondisi percabangan **IF** untuk memeriksa **nilai n**, apakah sama dengan **0** atau **1**. Jika **nilai n sama dengan 0** atau **n tidak sama dengan 1**, maka program akan **mengembalikan nilai n**. jika **nilai n tidak sama dengan 0** atau **n tidak sama dengan 1**, maka program akan **mengembalikan hasil dari fungsi Fibonacci(n-1) + Fibonacci(n-1)**. Di luar blok fungsi tersebut terdapat deklarasi variabel **x** yang bertipe **Integer** yang akan menerima inputan nilai dari keyboard. Kemudian terdapat perintah perulangan **for** dengan batas perulangan adalah nilai **x**. jika memenuhi batas dari syarat **nilai x**, maka program akan mencetak dan memanggil fungsi **fibonacci()** disertai dengan parameter berupa **variabel i** dan default end adalah kosong. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.30.



```
Masukan Batas Deret Bilangan Fibonacci : 10
Deret Fibonacci
0 1 1 2 3 5 8 13 21 34
```

Gambar 9.30 Hasil Keluaran Program

9.12 Fungsi Map (Map Function)

Fungsi `map()` digunakan untuk menerapkan suatu fungsi yang diberikan ke setiap item atau anggota iterable (List, Tuple, dan lain-lain) dan mengembalikannya dalam bentuk objek map. Objek `map()` ini nantinya dapat diubah ke bentuk list, tuple, dan lain-lain yang mana anggotanya adalah hasil dari proses anggota iterable lama.

Bentuk umum deklarasi fungsi `map()` adalah :

Map(function, iterable, ...)

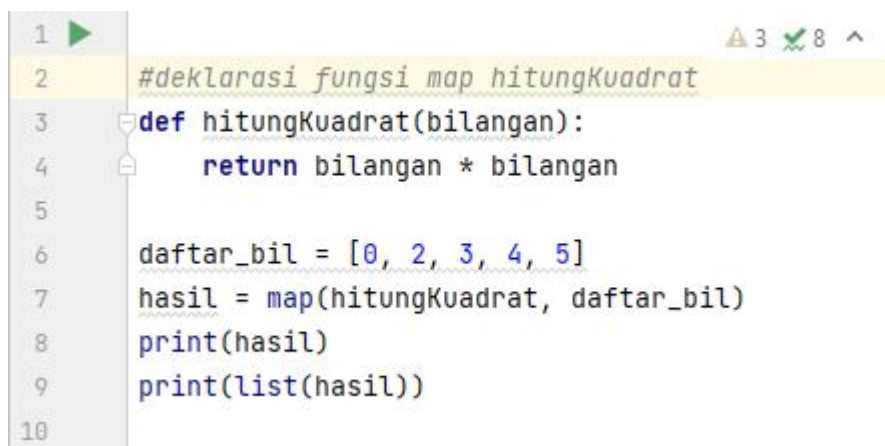
Fungsi `map()` membutuhkan dua parameter berupa:

- a. *Function* – fungsi yang dibutuhkan untuk memproses setiap item atau anggota iterable.
- b. *Iterable* – seperti List, Tuple, dan lain-lain yang nantinya akan diproses (dimapping) dengan fungsi pada argumen pertama.

Fungsi `map()` juga mengizinkan kita untuk meneruskan lebih dari satu iterable ke dalam parameter fungsi.

Studi Kasus 9.31

Program pada gambar 9.31 merupakan contoh penggunaan fungsi `map` pada file “`hitungKuadrat.py`”.



```
1  #deklarasi fungsi map hitungKuadrat
2
3  def hitungKuadrat(bilangan):
4      return bilangan * bilangan
5
6  daftar_bil = [0, 2, 3, 4, 5]
7  hasil = map(hitungKuadrat, daftar_bil)
8  print(hasil)
9  print(list(hasil))
10
```

Gambar 9.31 Contoh Penggunaan Fungsi Map

Perhatikan gambar 9.31, terdapat deklarasi fungsi dengan nama **"hitungKuadrat"**. Fungsi tersebut memiliki sebuah parameter, yaitu variabel **bilangan**. Terdapat perintah untuk mengembalikan hasil dari perkalian antara variabel **bilangan** dengan dirinya sendiri. Di luar blok fungsi tersebut terdapat variabel **daftar_bil** yang bertipe List dan sudah terisi nilai. Juga terdapat variabel **hasil** yang berisi nilai dari memanggil fungsi `map(hitungKuadrat, daftar_bil)`. Kemudian mencetak nilai dari variabel **hasil** dan `list(hasil)`. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.32.

```
<map object at 0x000001ACABF7B1C0>  
[0, 4, 9, 16, 25]
```

Gambar 9.32 Hasil Keluaran Program

9.13 Fungsi Filter (Filter Function)

Fungsi `filter()` digunakan untuk menyaring elemen, item, atau anggota iterable dengan bantuan fungsi yang bertugas menguji setiap anggota iterable apakah bernilai True atau False. Hasil atau nilai yang dikembalikan berupa iterable baru dari anggota iterable lama yang sudah melewati proses pengujian (dari fungsi yang dibuat) dan bernilai True.

Bentuk umum deklarasi fungsi `filter()` adalah :

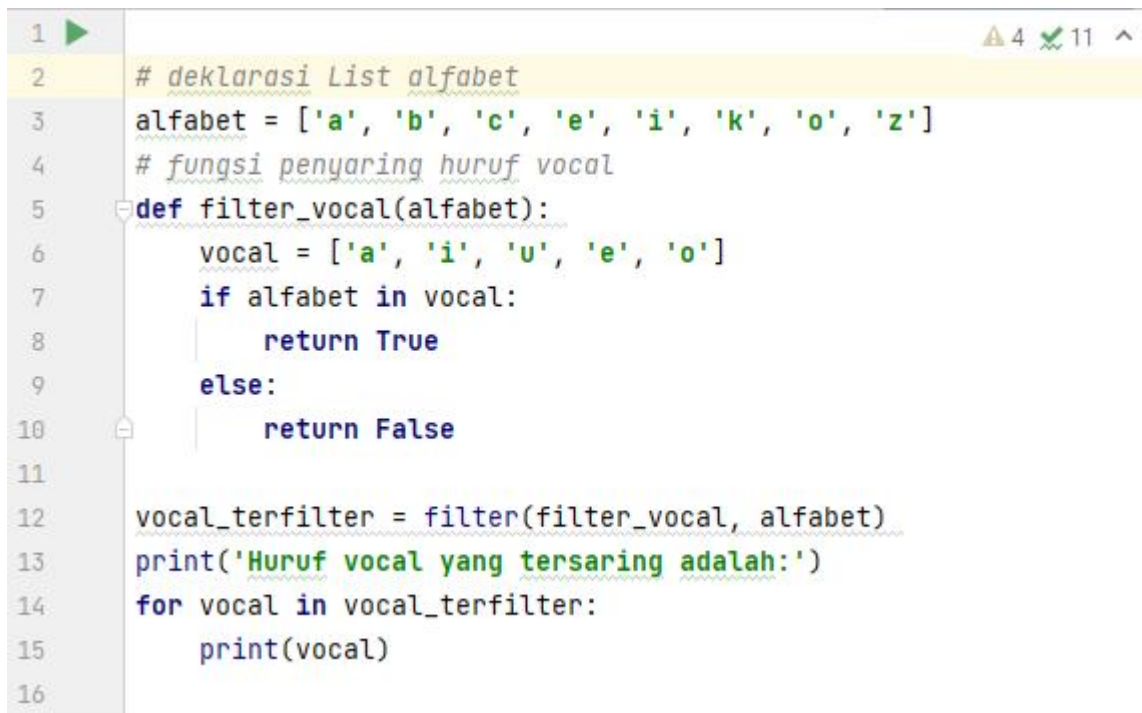
`filter(function, iterable, ...)`

Fungsi `filter()` membutuhkan dua parameter yaitu:

- *Function* – fungsi yang akan menguji apakah anggota atau elemen iterable bernilai True atau False. Jika tidak diisi, maka defaultnya adalah fungsi identitas yang akan mengembalikan nilai False, jika ada elemennya yang salah.
- *Iterable* – iterable yang akan difilter seperti list, tuple, dan lain-lain.

Studi Kasus 9.33

Program pada gambar 9.33 merupakan contoh penggunaan fungsi filter pada file “filter.py”.



```
1  # deklarasikan List alfabet
2  alfabet = ['a', 'b', 'c', 'e', 'i', 'k', 'o', 'z']
3  # fungsi penyaring huruf vocal
4  def filter_vocal(alfabet):
5      vocal = ['a', 'i', 'u', 'e', 'o']
6      if alfabet in vocal:
7          return True
8      else:
9          return False
10
11
12  vocal_terfilter = filter(filter_vocal, alfabet)
13  print('Huruf vocal yang tersaring adalah:')
14  for vocal in vocal_terfilter:
15      print(vocal)
16
```

Gambar 9.33 Contoh Penggunaan Fungsi Filter

Perhatikan gambar 9.33, terdapat deklarasi fungsi dengan variabel “**alfabet**” bertipe **List** dan sudah terisi data. Terdapat deklarasi fungsi **filter_vocal** dengan sebuah parameter, yaitu variabel **alfabet**. Di dalam fungsi tersebut, terdapat deklarasi variabel **vocal** bertipe **List** dan sudah terisi nilai. Terdapat kondisi percabangann IF untuk memeriksa nilai yang terdapat pada variabel **alpabet** dengan nilai yang terdapat pada variabel **vocal**. Jika kondisi **nilai alpabet terdapat pada variabel vocal**, maka akan mengembalikan **nilai True**. Jika kondisi **nilai alpabet tidak ada pada variabel vocal**, maka akan mengembalikan **nilai False**.

Di luar blok fungsi tersebut terdapat variabel **vocal_terfilter** yang berisi nilai dari memanggil fungsi **filter(filter_vocal, alfabet)**. Kemudian mencetak nilai yang terdapat pada variabel **vocal**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.34.

```
Huruf vocal yang tersaring adalah:  
a  
e  
i  
o
```

Gambar 9.32 Hasil Keluaran Program

9.14 Fungsi Anonim (Lambda Function)

Salah satu fitur Python yang cukup berguna adalah fungsi anonim. Dikatakan fungsi anonim karena fungsi ini tidak diberikan nama pada saat didefinisikan. Pada **fungsi biasa** kita menggunakan kata kunci **def** untuk membuatnya, sedangkan **fungsi anonim** ini kita menggunakan kata kunci **lambda**. Oleh karena itu, fungsi anonim ini dikenal juga dengan fungsi **lambda**. Bentuk umum deklarasi fungsi filter() adalah :

Lambda Arguments : expression

Fungsi lambda bisa mempunyai banyak argumen, tapi hanya boleh memiliki satu ekspresi. Ekspresi tersebutlah yang dikembalikan sebagai hasil dari fungsi. Fungsi lambda bisa kita simpan di dalam variabel untuk digunakan kemudian. Beberapa contoh kasus penerapan fungsi dapat dilihat pada contoh studi kasus dibawah ini.

Studi Kasus 9.33

Program pada gambar 9.33 merupakan contoh penggunaan fungsi lambda pada file "**Lambda1.py**".

```

1  ▶
2  # deklarasi fungsi lambda
3  # fungsi lambda dengan 1 argumen
4  kuadrat = lambda x: x*x
5  print("fungsi kuadrat lambda : ", kuadrat(3))
6
7  # fungsi lambda dengan 2 argumen
8  kali = lambda x, y: x*y
9  print("fungsi kali lambda : ", kali(4,3))
10

```

Gambar 9.33 Contoh Penggunaan Fungsi Lamda

Perhatikan gambar 9.33, terdapat deklarasi dua buah variabel, yaitu variabel **kuadrat** dan variabel **kali**. Variabel kuadrat digunakan untuk memanggil fungsi lambda pada variabel **x** dengan perintah **nilai x** dikalikan dirinya sendiri. Variabel kali digunakan untuk memanggil fungsi **lambda** pada **variabel x dan y**, dengan perintah **nilai x dikalikan nilai y**. Kemudian masing-masing variabel dipanggil dengan memberikan **nilai 3 pada fungsi kuadrat()**, serta memberikan **nilai 4 dan 3 pada fungsi kali**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.34.

```

fungsi kuadrat lambda :  9
fungsi kali lambda :  12

```

Gambar 9.34 Hasil Keluaran Program

Studi Kasus 9.35

Program pada gambar 9.35 merupakan contoh penggunaan fungsi lambda dan fungsi filter pada file "**Lambda2.py**".

```

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

```

Gambar 9.35 Contoh Penggunaan Fungsi Lamda dan Fungsi Filter

Perhatikan gambar 9.35, terdapat deklarasi dua buah variabel, yaitu variabel **my_list** dan variabel **list_ganjil**. Variabel **my_list** bertipe List dan sudah terisi nilai. Variabel **list_ganjil** digunakan untuk memanggil fungsi **list(filter(lambda()))** dengan perintah **mencari sisa bagi (modulus) dari nilai x setelah dibagi 2 tidak sama dengan 0**. Kemudian mencetak nilai dari variabel **list_ganjil**. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.36.

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

Gambar 9.37 Hasil Keluaran Program

Studi Kasus 9.38

Program pada gambar 9.38 merupakan contoh penggunaan fungsi lambda dan fungsi map pada file "**Lambda3.py**".

```

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

```

Gambar 9.38 Contoh Penggunaan Fungsi Lamda dan Fungsi Map

Perhatikan gambar 9.38, terdapat deklarasi dua buah variabel, yaitu variabel **my_list** dan variabel **kuadrar**. Variabel **my_list** bertipe List dan sudah terisi

nilai. Variabel kuadrat digunakan untuk memanggil fungsi `list(map(lambda() dengan perintah nilai x dikalikan dengan dirinya sendiri. Kemudian mencetak hasil dari variabel kuadrat. Adapun hasil keluaran dari program diatas ditampilkan dapat dilihat pada gambar 9.39.`

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

Gambar 9.39 Hasil Keluaran Program

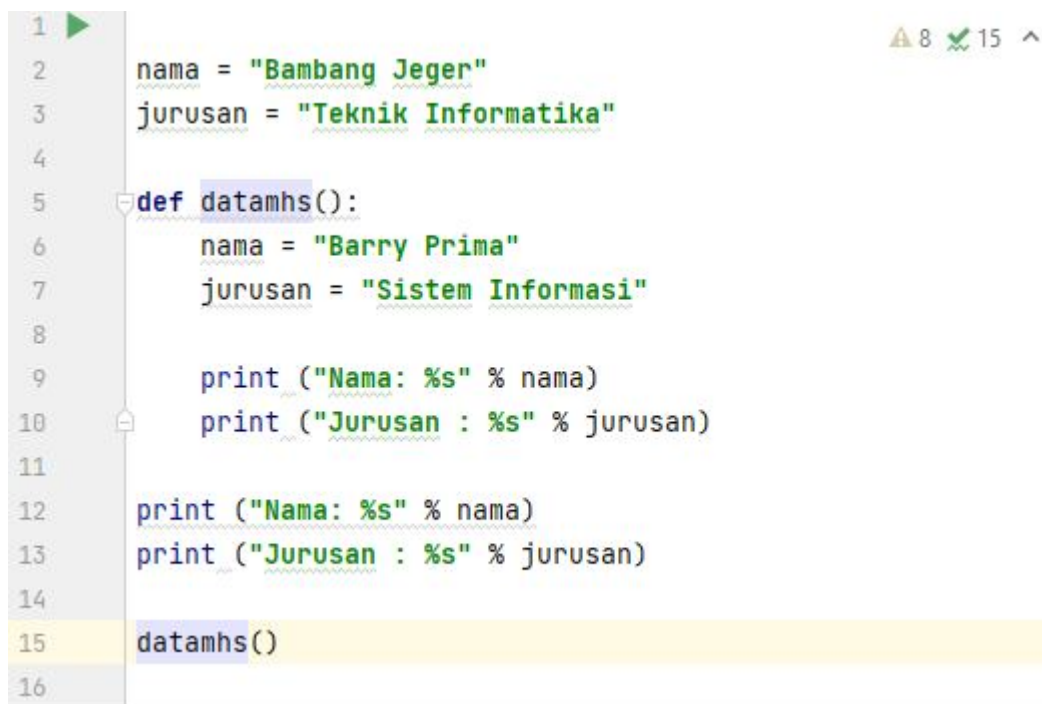
9.15 Praktikum

Langkah-langkah Praktikum

1. Buka Editor Python (IDLE / Pycharm / VSCode).
2. Buatlah file baru dengan membuka menu File > New > Source File atau dengan shortcut Ctrl + N.
3. Tulislah kode program berikut ini :

Program 9.1 : Praktikum91. Py

1. Buatlah program menggunakan bahasa Python untuk membuat prosedur datamhs sebagai berikut :



```
1  ▶
2  nama = "Bambang Jeger"
3  jurusan = "Teknik Informatika"
4
5  def datamhs():
6      nama = "Barry Prima"
7      jurusan = "Sistem Informasi"
8
9      print ("Nama: %s" % nama)
10     print ("Jurusan : %s" % jurusan)
11
12     print ("Nama: %s" % nama)
13     print ("Jurusan : %s" % jurusan)
14
15     datamhs()
16
```

Gambar 6.57 Kode Program Praktikum91

2. Simpan Program ini dengan nama Praktikum91.py
3. Jalankan program praktikum91 di atas, kemudian tuliskan apa yang tercetak pada variabel nama dan jurusan di dalam fungsi.

4. kemudian tuliskan apa yang tercetak pada variabel total di luar fungsi.

5. Jika kode program diatas diganti dengan kode program dibawah ini, apakah akan memiliki hasil keluaran yang sama? Jelaskan!

```
1  ▶ nama = "Bambang Jeger"
2  jurusan = "Teknik Informatika"
3
4
5  def datamhs():
6      nama = "Barry Prima"
7
8      print ("Nama: %s" % nama)
9      print ("Jurusan : %s" % jurusan)
10
11     print ("Nama: %s" % nama)
12     print ("Jurusan : %s" % jurusan)
13
14     datamhs()
15
```

Gambar 6.58 Modifikasi Kode Program Praktikum91

Program 9.2 : Praktikum92. Py

1. Buatlah program menggunakan bahasa Python untuk membuat fungsi penjumlahan sebagai berikut :

```

#Deklarasi Fungsi Operator
def fungsi_total_nilai(var_harian, var_uts, var_uas):
    var_harian = int(var_harian) * 0.3
    var_uts = int(var_uts) * 0.3
    var_uas = int(var_uas) * 0.4

    var_total = var_harian + var_uts + var_uas
    return var_total

#Deklarasi Fungsi Percabangan
def fungsi_percabangan(var_nilai):
    var_huruf = ""
    if (var_nilai >= 0 and var_nilai < 20):
        var_huruf = "E"
    elif (var_nilai >= 20 and var_nilai < 40):
        var_huruf = "D"
    elif (var_nilai >= 40 and var_nilai < 60):
        var_huruf = "C"
    elif (var_nilai >= 60 and var_nilai < 80):
        var_huruf = "B"
    elif (var_nilai >= 80 and var_nilai < 100):
        var_huruf = "A"
    return var_huruf

#Deklarasi Fungsi Perulangan
def fungsi_perulangan():
    var_hasil_perulangan = 0
    for i in range(1,3):
        print("-----Nilai Ke " + str(i) + "-----")
        var_harian = input("Nilai Tugas : ")
        var_uts = input("Nilai UTS : ")
        var_uas = input("Nilai UAS : ")

        #Pemanggilan fungsi Penjumlahan
        var_hasil_perulangan += (int(fungsi_total_nilai(var_harian, var_uts, var_uas)))

    return var_hasil_perulangan

#Pemanggilan fungsi perulangan
var_total = fungsi_perulangan()

print("-----Total Nilai -----")
print("Total nilai yang didapat : " + str(var_total))

#Pemanggilan Fungsi Percabangan
print("Total Nilai Dalam Huruf : ", fungsi_percabangan(var_total))

```

Gambar 6.59 Kode Program Praktikum92

2. Simpan Program ini dengan nama Praktikum92.py
3. Jalankan program praktikum91 di atas, kemudian tuliskan apa yang tercetak pada var_total di dalam fungsi.

4. kemudian tuliskan apa yang tercetak dari hasil fungsi_percabangan(var_total) di luar fungsi.

9.16 Rangkuman

1. Prosedur adalah blok / subprogram yang berisi perintah / aksi untuk mengerjakan tugas tertentu dan tidak mengembalikan nilai pada saat nama prosedur dipanggil.
2. Fungsi adalah blok / subprogram yang berisi perintah / aksi untuk mengerjakan tugas tertentu dan mengembalikan nilai pada saat nama fungsi dipanggil.
3. Fungsi yang dibuat berdasarkan kebutuhan pengguna disebut dengan **User-Defined Functions**. Sedangkan fungsi yang sudah siap untuk digunakan dan merupakan bawaan dari bahasa pemrograman Python disebut **Built-In Functions**.
4. Fungsi lanjutan dalam bahasa pemrograman Python adalah fungsi Rekursif, fungsi Filter, Fungsi Map dan Fungsi Anonim (Lambda).
5. Fungsi Rekursif digunakan jika ingin membuat fungsi yang dapat memanggil dirinya sendiri secara berulang-ulang hingga suatu kondisi yang didefinisikan terpenuhi atau bernilai benar (True).
6. Fungsi Filter digunakan digunakan untuk menyaring elemen, item, atau anggota iterable dengan bantuan fungsi yang bertugas menguji setiap anggota iterable apakah bernilai True atau False.
7. Fungsi Map digunakan untuk untuk mengaplikasikan satu fungsi ke semua anggota dari iterable (list, tuple, dan lain – lain) dan mengembalikan hasilnya berupa objek map.
8. Fungsi Anonim (Lambda) adalah fungsi anonim (fungsi tanpa nama) yang dapat ditugaskan ke variabel atau yang dapat dilewatkan sebagai argumen ke fungsi lain.

9.17 Latihan

1. Buatlah program Python untuk menghitung luas persegi Panjang, segita dan lingkaran dengan menggunakan prosedur dalam satu file.
2. Buatlah program Python untuk menghitung luas persegi Panjang, segitiga dan lingkaran dimana setiap prosedur disimpan di dalam file yang berbeda.

3. Buatlah program untuk mencetak daftar nama mahasiswa yang terdiri atas NIM, Nama Lengkap, Jurusan dan Telepon. Nilai yang diinput adalah banyaknya data (N) dan data masing-masing mahasiswa. Program terdiri dari tiga prosedur, yaitu : Inputdata (N), Proses (X) dan CetakHasil. Adapun keluaran program terlihat seperti gambar dibawah ini :

NIM	Nama Lengkap	Jurusan	Telepon
XXX	XXX	XXX	XXX
XXX	XXX	XXX	XXX
XXX	XXX	XXX	XXX

9.18 Tugas Mandiri

Kerjakan soal-soal berikut :

1. Buatlah program untuk menampilkan bilangan kelipatan X saja dari sekelompok data menggunakan List. Nilai yang diinput adalah banyaknya data (N) dan data masing-masing (dalam bentuk List / Daftar). Program terdiri dari tiga prosedur, yaitu : Inputdata (N), Proses (X) dan CetakHasil. Adapun keluaran program terlihat seperti gambar dibawah ini :

No. Indeks	Bilangan Genap
XXX	XXX
XXX	XXX
XXX	XXX

2. Buatlah program untuk menjumlahkan data antara dua buah list dengan menggunakan fungsi, jika diketahui List A = [1, 2, 4, 8] dan List B = [16, 32, 64, 128].
3. Buatlah program untuk menampilkan jumlah deret Aritmatika yang masukannya adalah suku awal, beda dan banyaknya suku. Definisikan fungsi Aritmatika(A, B, N) dimana A untuk suku awal, B untuk beda dan N untuk banyaknya suku untuk menyelesaikan persoalan ini.



FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS BUDI LUHUR

Jl. Raya Ciledug, Petukangan Utara, Pesanggrahan
Jakarta Selatan, 12260
Telp: 021-5853753 Fax : 021-5853752
<http://fti.budiluhur.ac.id>