

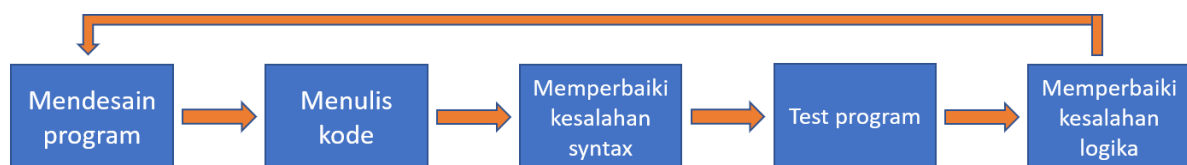
Input dan Output

OBJEKTIF:

1. Mahasiswa mampu membuat program dengan Python.
2. Mahasiswa mampu memahami konsep input.
3. Mahasiswa mampu memahami konsep output.
4. Mahasiswa mampu memahami tata cara menulis program.
5. Mahasiswa mampu memahami fungsi pada Python.
6. Mahasiswa mampu memahami *Module Standard Library* pada Python.

2.1 Membuat Program

Membuat sebuah program umumnya membutuhkan lima fase pekerjaan seperti terlihat pada diagram berikut:



1. Mendesain program

Ketika programmer membuat sebuah program, mereka tidak langsung menulis kode namun memulainya dengan mendesain program. Mendesain program melibatkan tiga tahap: menganalisa persoalan yang ingin diselesaikan, menentukan spesifikasi (input dan output program), dan membuat langkah-langkah detil dari program.

2. Menulis kode

Setelah mendapatkan desain program, programmer memulai menulis kode dalam bahasa pemrograman *high-level* seperti Python.

3. Memperbaiki kesalahan syntax

Kesalahan penulisan syntax, seperti lupa menggunakan tanda kutip, atau salah pengejaan akan menyebabkan kode program tidak dapat dijalankan. Hampir semua program yang ditulis pertama kali akan terdapat error syntax. Kita harus memperbaiki error syntax ini supaya program dapat dijalankan.

4. Test Program

Setelah kode program dapat berjalan, program tersebut harus diuji untuk mencari apakah terdapat error logika. Error logika adalah kesalahan yang tidak mencegah program berjalan, namun menghasilkan hasil yang tidak sesuai. Error logika sering disebut dengan bug.

5. Memperbaiki kesalahan logika

Jika program menghasilkan kesalahan logika, programmer melakukan debugging kode.

Debugging adalah proses mencari dan memperbaiki error logika. Jika programmer menemukan bahwa desain awal program harus diubah, fase pengembangan program diulang ke mendesain program.

2.1.1 Proses Desain Suatu Program

Proses mendesain program dapat dirangkum menjadi tiga langkah:

- **Menganalisa persoalan.** Kita harus memahami persoalan yang ingin diselesaikan oleh program kita.
- **Menentukan spesifikasi.** Kita harus menentukan apa yang akan dicapai oleh program kita. Misalkan untuk program sederhana, kita menentukan input dan output dari program dan bagaimana kaitan input dan output tersebut
- **Menentukan langkah-langkah penyelesaian.** Dari spesifikasi program, kita membuat langkah-langkah yang harus dilakukan oleh program untuk menyelesaikan persoalan tersebut

Kita mendesain program dengan menjabarkan langkah-langkah detil terurut yang harus dilakukan program tersebut untuk menyelesaikan tugasnya. Langkah-langkah detil terurut ini disebut dengan **algoritma**. Kita dapat membayangkan algoritma seperti sebuah resep.

Misalkan algoritma untuk membuat teh dapat dituliskan seperti berikut:

1. Masukkan teh celup ke cangkir
2. Isi teko dengan air
3. Masak air dalam teko sampai mendidih
4. Tuangkan air mendidih ke cangkir
5. Tambahkan gula
6. Aduk teh

Terdapat dua alat bantu yang digunakan untuk menuliskan algoritma, yaitu **pseudocode** dan **flowchart**:

- **Pseudocode**

Pseudo berarti semu atau tidak nyata. **Pseudocode** (kode yang tidak nyata) adalah penjelasan langkah-langkah dari suatu algoritma yang menggunakan bahasa informal dan tidak mengikuti aturan syntax dari bahasa pemrograman tertentu. Misalkan kita diminta untuk membuat sebuah program yang mengkonversi temperatur dalam fahrenheit ke temperatur dalam celsius, kita dapat menuliskan algoritma program dalam pseudocode seperti berikut:

```
Input temperatur dalam fahrenheit
kalkulasi konversi fahrenheit ke celcius dengan rumus:
    celcius = (5/9) * (fahrenheit - 32)
Tampilkan temperatur dalam celcius
```

- **Flowchart**

Flowchart adalah diagram yang menjelaskan langkah-langkah dari algoritma. Berikut komponen-komponen dari flowchart:



Oval

Menandakan awal atau akhir



Jajaran Genjang

Menandakan input atau output



Persegi Panjang

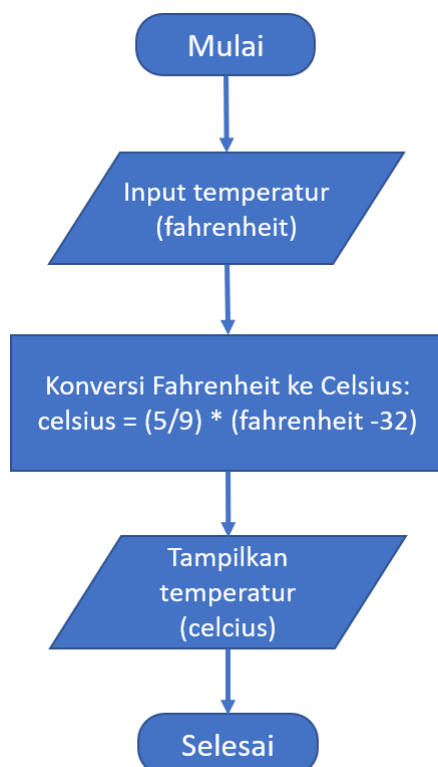
Menandakan proses



Panah

Menandakan alur

Gambar di bawah adalah contoh flowchart dari algoritma konversi temperatur fahrenheit ke celsius:



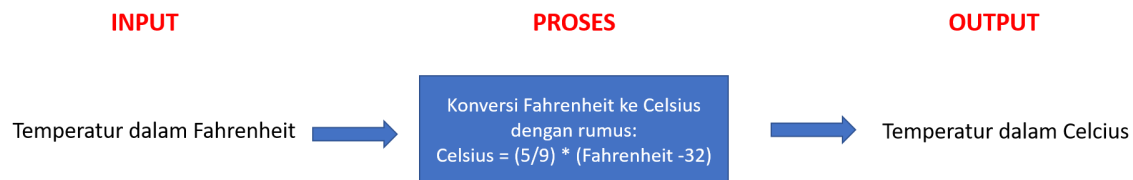
2.1.2 Input, Proses, dan Output

Sebuah program umumnya melakukan tiga hal berikut:

- Menerima Input
- Memroses input
- Menghasilkan Output

Input adalah data yang program terima saat berjalan. Satu bentuk input yang paling umum adalah data yang diketikkan pada keyboard. Setelah input diterima, program melakukan pemrosesan input seperti melakukan kalkulasi matematika terhadap input. Hasil dari pemrosesan kemudian dikeluarkan sebagai **output**. Satu bentuk output yang paling umum adalah menampilkan teks ke layar.

Gambar berikut mengilustrasikan Input, Proses, dan Output dari sebuah program yang mengkonversi fahrenheit dan celcius:



2.2 Input

Python menyediakan fungsi built-in `input()` yang digunakan untuk membaca input dari keyboard. Fungsi `input()` adalah fungsi yang mempunyai side-effect dan mengembalikan sebuah nilai. Fungsi `input()` menerima satu argumen berupa sebuah string untuk ditampilkan ke layar sebagai *prompt* (pesan ke pengguna bahwa program siap menerima input) dan mengembalikan semua yang diketikkan pengguna sebagai string. Kita umumnya menggunakan fungsi `input()` dalam statement assignment seperti berikut:

```
<variabel> = input(<prompt>)
```

Berikut adalah sebuah contoh statement yang menerima input dari keyboard:

```
nama = input('Masukkan nama Anda: ')
```

Ketika statement di atas dieksekusi:

- Teks 'Masukkan nama Anda: ' ditampilkan ke layar
- Program berhenti dan menunggu pengguna untuk mengetikkan sesuatu pada keyboard sampai mengakhirinya dengan menekan Enter
- Ketika tombol Enter ditekan, data yang diketikkan dikembalikan sebagai sebuah string dan ditugaskan ke variabel nama

Berikut adalah sesi interaktif yang mencontohkan penggunaan fungsi `input()`:

Karakter-karakter yang diketikkan pengguna dituaskan ke variable nama

```
>>> nama = input('Masukkan nama Anda: ')
Masukkan nama Anda: Budi
>>> print(nama)
Budi
```

Diketikkan pengguna

2.2.1 Membaca Angka dengan Fungsi `input()`

Fungsi `input()` selalu mengembalikan string, meskipun pengguna mengetikkan angka:

```
>>> n = input('Masukkan sebuah angka: ')
Masukkan sebuah angka: 50
>>> type(n)
<class 'str'> # Tipe data dari variabel yang menyimpan input adalah string
```

Untuk mendapatkan input berupa tipe numerik, kita harus mengkonversi nilai kembali fungsi input dengan fungsi `int()` untuk konversi ke integer atau dengan fungsi `float()` untuk konversi ke floating point. Sesi interaktif berikut mencontohkan penggunaan fungsi konversi `int()` untuk mengkonversi input dari pengguna ke integer:

```
>>> num = input('Masukkan sebuah angka: ')
Masukkan sebuah angka: 50
>>> n = int(num) # konversi ke integer dan simpan ke hasil konversi ke variabel
n
>>> type(n)
<class 'int'>
```

Kita dapat langsung mengkonversi nilai kembali input ke tipe numerik dalam satu statement. Dua statement berikut:

```
num = input('Masukkan sebuah angka: ')
n = int(num)
```

dapat dituliskan dalam satu baris statement menjadi seperti berikut:

```
num = int(input('Masukkan sebuah angka: '))
```

Untuk mendapatkan input pengguna berupa tipe floating point kita menggunakan fungsi `float()`. Sesi interaktif berikut mencontohkan cara mendapatkan input tipe floating point:

```
>>> jarak = float(input('Masukkan jarak dalam m: '))
Masukkan jarak dalam m: 34.7
>>> print(jarak)
34.7
```

Fungsi `int()` dan `float()` hanya bekerja jika string yang dikonversi hanya terdiri dari karakter-karakter angka. Jika argumen dari fungsi `int()` dan `float()` tidak dapat dikonversi, sebuah error eksepsi akan muncul:

```
>>> num = int(input('Masukkan sebuah angka: '))
Masukkan sebuah angka: xyz
Traceback (most recent call last):
  File "<pyshe11#68>", line 1, in <module>
    num = int(input('Masukkan sebuah angka: '))
ValueError: invalid literal for int() with base 10: 'xyz'
```

Eksepsi adalah error yang terjadi saat program berjalan yang menyebabkan program crash (berhenti).

2.3 Output

Kita menampilkan output ke layar menggunakan fungsi `print()`. Sebelumnya kita telah melihat cara menggunakan fungsi `print()` dengan satu argumen, sebenarnya fungsi `print()` dapat digunakan dengan argumen lebih dari satu:

```
>>> nama_depan = 'Budi'
>>> nama_belakang = 'Susilo'
>>> print('Nama Lengkap:', nama_depan, nama_belakang)
Nama Lengkap: Budi Susilo
```

Dengan argumen lebih dari satu, fungsi `print()` menampilkan masing-masing nilai argumen dengan dipisahkan spasi.


Argumen dari fungsi `print()` juga dapat berupa ekspresi:

```
>>> x = 34
>>> print('Nilai x = ', x)
Nilai x = 34
>>> print('Nilai x * 10 =', (x*10))
Nilai x * 10 = 340
```

2.3.1 Menentukan Pemisah pada Tampilan `print()`

Kita dapat mengganti pemisah antara tampilan nilai-nilai argumen dari fungsi `print()` dengan menambahkan argumen keyword `sep` (singkatan dari separator):


```
>>> print('Satu', 'Dua', 'Tiga', sep='*')
Satu*Dua*Tiga
```



Argumen keyword `sep` dituliskan dengan `sep=<string>`.
`<string>` akan ditampilkan sebagai pemisah nilai-nilai argumen

Nilai dari argumen keyword `sep` dapat berupa string apapun. Berikut adalah contoh mengganti pemisah antara tampilan teks-teks argument `print()` dengan string `'Pemisah'`:

```
>>> print('Satu', 'Dua', 'Tiga', sep='Pemisah')
SatuPemisahDuaPemisahTiga
```



Fungsi `print()` juga mempunyai argumen keyword `end` yang digunakan untuk menentukan pengakhir dari tampilan:

```
>>> print('a', 'b', 'c', sep=',', end='!')
a,b,c!
```



Argumen keyword `end` dituliskan dengan `end=<string>`.
`<string>` akan ditampilkan sebagai pengakhir tampilan nilai-nilai argumen

Sama seperti argument keyword `sep`, nilai dari argumen keyword `end` dapat berupa string apapun:

```
>>> print('a', 'b', 'c', sep=',', end='<akhir dari output>')
a,b,c<akhir dari output>
```

Argumen keyword `end` dituliskan dengan `end=<string>`.
`<string>` akan ditampilkan sebagai pengakhir tampilan nilai-nilai argumen

2.3.2 Escape Sequence

Escape sequence adalah kombinasi karakter `\` dengan sebuah karakter tertentu yang digunakan sebagai alternatif penulisan karakter tertentu tersebut. Salah satu contoh escape sequence adalah `\n`. Escape sequence `\n` digunakan untuk mencetak baris baru:

```
>>> print('Satu\nDua\nTiga')
Satu
Dua
Tiga
```

'`\n`' menyebabkan karakter-karakter
setelahnya dicetak dalam baris baru

Tabel berikut mendaftar escape sequence yang sering digunakan:

Escape Sequence	Keterangan
<code>\"</code>	Digunakan sebagai alternatif penulisan tanda kutip ganda
<code>\'</code>	Digunakan sebagai alternatif penulisan tanda kutip tunggal
<code>\\</code>	Digunakan sebagai alternatif penulisan backslash <code>\</code>
<code>\t</code>	Digunakan untuk mencetak tab (indentasi)
<code>\n</code>	Digunakan untuk mencetak baris baru

Sesi-sesi interaktif berikut mencontohkan penggunaan escape sequence:

```
>>> print('satu\tdua\ntiga\tempat')
satu    dua
tiga    empat
```

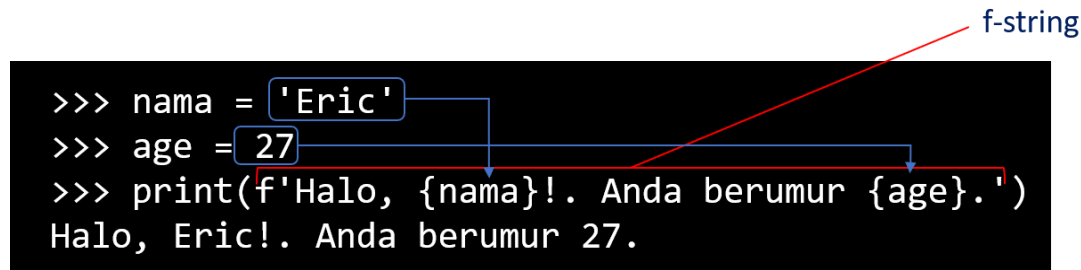
```
>>> print('Simpan file Anda ke C:\\temp\\data')
Simpan file Anda ke C:\\temp\\data
```

```
>>> print('Budi berkata \"Saya pergi hari Jum\\'at kemarin\"')
Budi berkata "Saya pergi hari Jum'at kemarin"
```

2.3.3 Memformat Output dengan f-string

Python menyediakan f-string (formatted string) yang memudahkan kita untuk memformat tampilan output:

```
>>> nama = 'Eric'
>>> age = 27
>>> print(f'Halo, {nama}!. Anda berumur {age}.')
Halo, Eric!. Anda berumur 27.
```



Kita menggunakan f-string dengan menuliskan awalan `f` pada suatu string dan menuliskan variabel atau ekspresi di dalam tanda kurawal. Berikut adalah contoh penggunaan f-string untuk menampilkan nilai dari variabel `nama` dan nilai dari ekspresi `age + 5`:

```
>>> print(f'{nama} akan berumur {age + 5}, 5 tahun mendatang.')
Eric akan berumur 32, 5 tahun mendatang.
```

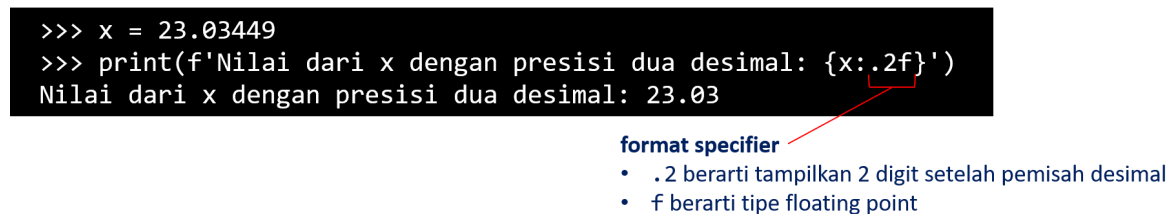
Memformat Output Floating Point

Ketika kita menggunakan fungsi `print()` untuk menampilkan nilai floating point, banyak digit setelah pemisah desimal yang tampil bisa sampai dengan 12 digit:

```
>>> hasil = 1/3
>>> print(hasil)
0.3333333333333333
```

Kita dapat menentukan berapa digit setelah pemisah desimal yang ditampilkan dengan menggunakan **format specifier**. Sesi interaktif berikut mencontohkan penggunaan format specifier:

```
>>> x = 23.03449
>>> print(f'Nilai dari x dengan presisi dua desimal: {x:.2f}')
Nilai dari x dengan presisi dua desimal: 23.03
```



- `.2` berarti tampilkan 2 digit setelah pemisah desimal
- `f` berarti tipe floating point

Menambahkan tanda koma sebelum titik pada format specifier menyebabkan digit sebelum pemisah desimal ditampilkan dengan pemisah ribuan:


```
>>> harga = 12577500
>>> print(f'Harga per unit setelah PPN: Rp.{harga * 1.1:,.2f}')
Harga per unit setelah PPN: Rp.13,835,250.00
```

Tanda koma untuk menampilkan pemisah ribuan pada digit-digit sebelum pemisah desimal

Kita juga dapat menentukan lebar jumlah digit tampilan:

```
>>> saldo = 25047.50
>>> print(f'Saldo rekening Anda: Rp.{saldo:12,.2f}')
Saldo rekening Anda: Rp. 25,047.50
```

Total karakter nilai floating point (termasuk koma dan titik) adalah 9 karakter. Jadi tampilan ditambahkan 3 spasi di depan untuk mencapai 12 karakter minimum

Angka 12 berarti tampilan dengan minimum lebar 12 karakter

Memformat Tampilan Integer

Format specifier dapat digunakan juga untuk memformat tampilan integer:

```
>>> populasi = 270203917
>>> print(f'Populasi Indonesia tahun 2020: {populasi: ,d} jiwa.')
Populasi Indonesia tahun 2020: 270,203,917 jiwa.
```

Format specifier untuk integer:
• d berarti menampilkan tipe integer

Hanya pemisah ribuan dan lebar minimum yang dapat dituliskan pada format specifier integer (kita tidak bisa menuliskan presisi):

```
>>> stok = 1473
>>> print(f'Stok barang: {stok: 12,d} unit.')
Stok barang: 1,473 unit.
```

2.4 Menulis Program

Pseudocode algoritma konversi temperatur fahrenheit ke celcius pada bagian sebelumnya:

```
Input temperatur dalam fahrenheit
kalkulasi konversi fahrenheit ke celcius dengan rumus:
    celcius = (5/9) * (fahrenheit - 32)
Tampilkan temperatur dalam celcius
```

dapat dituliskan dalam kode program Python menjadi seperti berikut:

```
fahrenheit = float(input('Masukkan temperatur dalam Fahrenheit: '))
celcius = 5/9 * (fahrenheit - 32)
print(f'Temperatur dalam Celcius adalah {celcius:.2f} derajat Celcius')
```

Untuk menjalankan program di atas, kita pertama-tama harus mengetikkannya pada modul script IDLE, lalu menyimpannya dalam sebuah file dengan nama `konversi_suhu.py` (sesuai konvensi program Python disimpan dalam sebuah file dengan ekstensi `.py`).

Jalankan program `konversi_suhu.py` dan uji dengan memasukkan angka 89, Anda akan mendapatkan output seperti berikut:

```
Masukkan temperatur dalam Fahrenheit: 89
Temperatur dalam Celcius adalah 31.67 derajat Celcius
```

Ketika sebuah program dijalankan, interpreter Python mengeksekusi statement-statement pada file program secara berurut dari baris paling atas hingga baris paling bawah.

2.4.1 Comment

Comment adalah catatan pendek yang dituliskan di berbagai bagian program untuk menjelaskan cara kerja bagian tersebut. Kita menuliskan comment untuk menjelaskan cara kerja program kita ke programmer lain atau sebagai pengingat ke diri kita.

Kita menulis comment dengan mengawali teks dengan karakter `#`:

```
# Program untuk mengkonversi Fahrenheit ke Celcius
fahrenheit = float(input('Masukkan temperatur dalam Fahrenheit: '))
celcius = 5/9 * (fahrenheit - 32)
print(f'Temperatur dalam Celcius adalah {celcius:.2f} derajat Celcius')
```

Baris pertama program di atas adalah comment. Semua teks setelah `#` sampai dengan akhir baris diabaikan oleh interpreter sehingga tidak mempunyai efek apapun ke program.

Kita juga dapat menuliskan comment setelah statement seperti berikut:

```
celcius = 5/9 * (fahrenheit - 32) # rumus konversi fahrenheit ke celcius
```

2.4.2 Kesalahan Penulisan Syntax

Ketika kita menjalankan program Python, sebelum mengeksekusi kode-kode di dalam program, interpreter Python akan memeriksa apakah kode dalam program tersebut mengikuti syntax-syntax yang benar. Jika terdapat kesalahan penulisan syntax, interpreter akan menampilkan pesan error yang disebut dengan *Syntax Error* dan menghentikan eksekusi kode-kode di dalam program. Sebagai contoh, misalkan kita lupa menuliskan tanda kutip pada argumen fungsi `print()` di baris 4 pada program `konversi_suhu.py`. Ketika kita mencoba menjalankan program tersebut, kita akan mendapatkan pesan *Syntax Error* seperti terlihat pada gambar berikut:

```
# Program untuk mengkonversi temperatur Fahrenheit ke Celcius
fahrenheit = float(input('Masukkan temperatur dalam Fahrenheit: '))
celcius = 5/9 * (fahrenheit - 32) # rumus konversi fahrenheit ke celcius
print(f'Temperatur dalam Celcius adalah {celcius:.2f} derajat Celcius)
```

```
File "c:/ilab/konversi_suhu.py", line 4
    print(f'Temperatur dalam Celcius adalah {celcius:.2f} derajat Celcius)
                                             ^
```

```
SyntaxError: EOL while scanning string literal
```

Terjadi kesalahan syntax

Kesalahan ada di baris 4

EOL berarti End Of Line (akhir baris).
Interpreter tidak menemukan penanda
akhir string sampai akhir baris

2.5 Fungsi

Fungsi adalah kelompok statement-statement yang diberikan sebuah nama. Tujuan membuat fungsi:

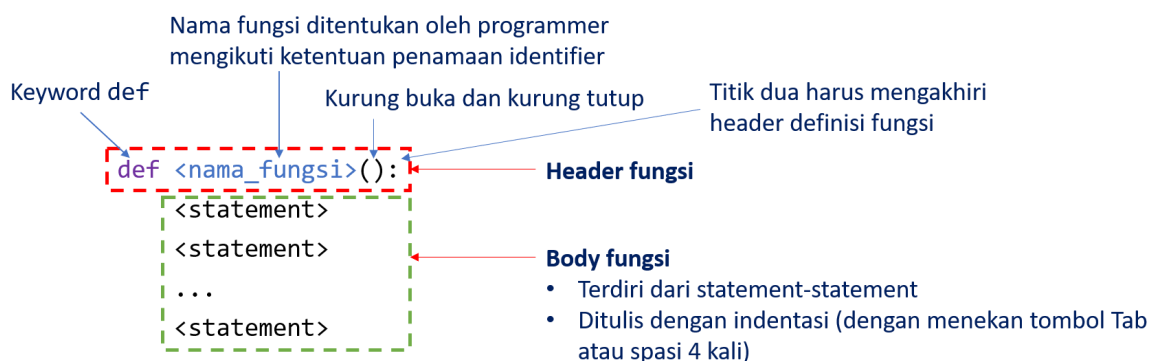
- *Reusability* (Penggunaan Ulang) - bagian kode-kode dari program yang umumnya digunakan berulang dapat dituliskan dalam sebuah fungsi, sehingga dapat mengurangi penulisan kode yang berulang
- *Modularity* (Modularitas) - memecah logika-logika program menjadi bagian-bagian kecil sehingga program mudah dibaca dan dikelola

2.5.1 Menulis Fungsi

Untuk membuat fungsi, kita menuliskan **definisi fungsi**. Bentuk dasar syntax penulisan definisi fungsi:

```
def <nama_fungsi>():  
    <statement>  
    <statement>  
    ...  
    <statement>
```

Gambar berikut menjelaskan bagian-bagian dari definisi fungsi:



Perhatikan bahwa statement-statement di dalam body fungsi dituliskan menjorok ke dalam (diindentasi). Kita mengindentasi dengan menekan tombol `Tab` atau dengan mengetikkan spasi sebanyak 4 kali.

Berikut adalah contoh definisi fungsi bernama `pesan`:

```
def pesan():  
    print('Selamat Datang!')  
    print('Kita akan mempelajari Bahasa Python.')
```

Jika kita menyimpan kode definisi fungsi di atas ke dalam sebuah file `cetak_pesan.py` dan mencoba menjalankan file tersebut, kita tidak akan mendapatkan output apapun. Ini karena ketika interpreter membaca definisi fungsi, interpreter tidak mengeksekusi statement-statement pada body fungsi, namun menyimpannya dalam memori.

Untuk mengeksekusi statement-statement pada body fungsi, kita harus memanggil fungsi tersebut. Kita memanggil fungsi `pesan`, dengan menuliskan statement `pesan()`. Sesi interaktif berikut mencontohkan pemanggilan fungsi `pesan` setelah file `cetak_pesan.py` dijalankan:

```
>>> pesan()
Selamat Datang!
Kita akan mempelajari Bahasa Python.
```

Ketika kita memanggil sebuah fungsi, interpreter akan mengeksekusi statement-statement dalam body fungsi tersebut dari statement pertama (paling atas) sampai dengan statement terakhir (paling bawah).

Kode yang kita tuliskan pada file `cetak_pesanan.py` sebelumnya hanyalah sebuah definisi fungsi dan bukanlah program lengkap. Untuk menjadikannya sebuah program lengkap, kita harus menambahkan pemanggilan fungsi `pesan` pada baris terakhir dari file `cetak_pesanan.py`:

```
# cetak_pesanan.py
# Program ini mendemonstrasikan sebuah fungsi

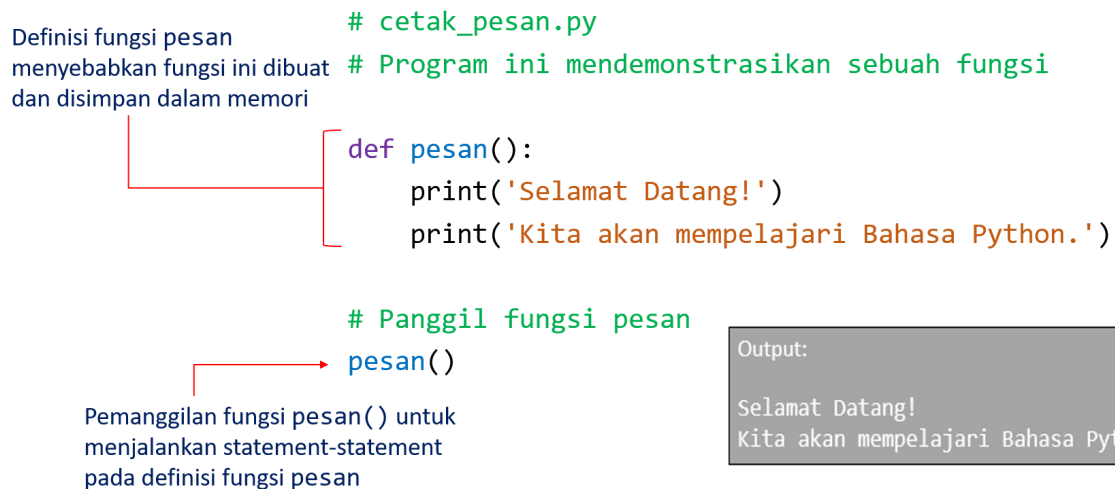
def pesan():
    print('Selamat Datang!')
    print('Kita akan mempelajari Bahasa Python.')

# Panggil fungsi pesan
pesan()
```

Jika kita menjalankan file `cetak_pesanan.py` yang berisi kode di atas, kita akan mendapatkan output berikut:

```
Selamat Datang!
Kita akan mempelajari Bahasa Python.
```

Gambar berikut menjelaskan bagian-bagian kode program pada file `cetak_pesanan.py`:



Kita membuat fungsi dengan **menuliskan definisi fungsi** dan untuk menggunakannya kita harus melakukan **memanggil fungsi** tersebut. Gambar berikut mengilustrasikan definisi fungsi dan pemanggilan fungsi:

Definisi Fungsi

```
def cetak_pesan():  
    print('Selamat Datang!')  
    print('Ada yang bisa kami bantu?')
```

Pemanggilan Fungsi

```
cetak_pesan()
```

2.5.2 Fungsi `main()`

Kita dapat menuliskan lebih dari satu definisi fungsi dalam sebuah program. Sebuah program umumnya mempunyai sebuah fungsi yang berisi logika utama dari program. Fungsi ini umumnya dinamakan sebagai fungsi `main`. Fungsi `main` ini yang dijalankan ketika program dimulai dan memanggil fungsi-fungsi lain ketika diperlukan. Sebuah program lengkap umumnya dimulai dengan kerangka seperti berikut:

```
def main():  
    # Statement-statement logika utama program  
    #...  
  
# Panggil fungsi main  
main()
```

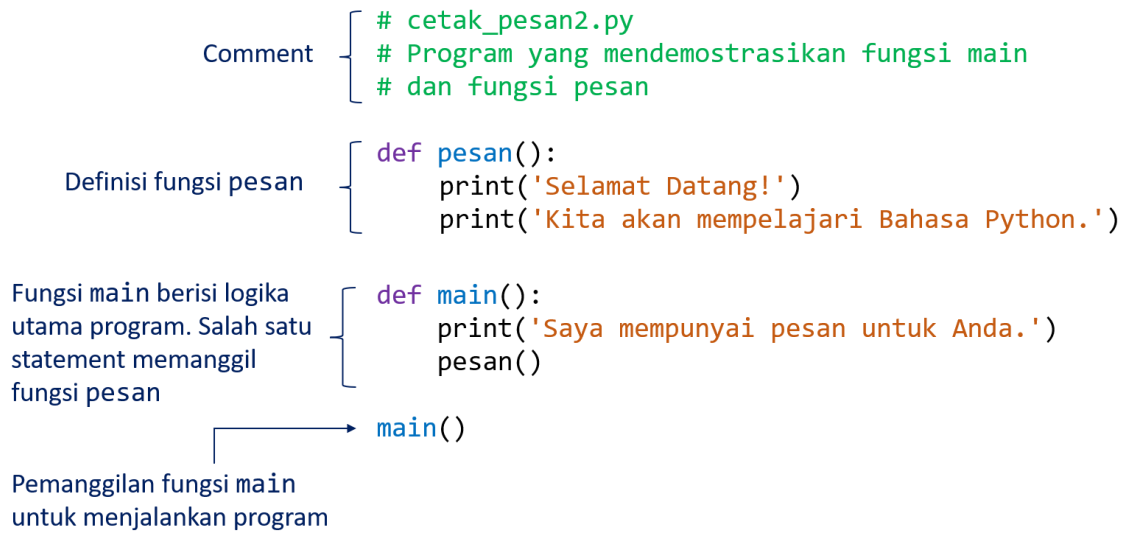
Contoh program yang mendemonstrasikan fungsi `main` yang memanggil fungsi `pesan`:

```
# cetak_pesan2.py  
# Program yang mendemonstrasikan fungsi main  
# dan fungsi pesan  
  
def pesan():  
    print('Selamat Datang!')  
    print('Kita akan mempelajari Bahasa Python.')  
  
def main():  
    print('Saya mempunyai pesan untuk Anda.')  
    pesan() # memanggil fungsi pesan  
  
main()
```

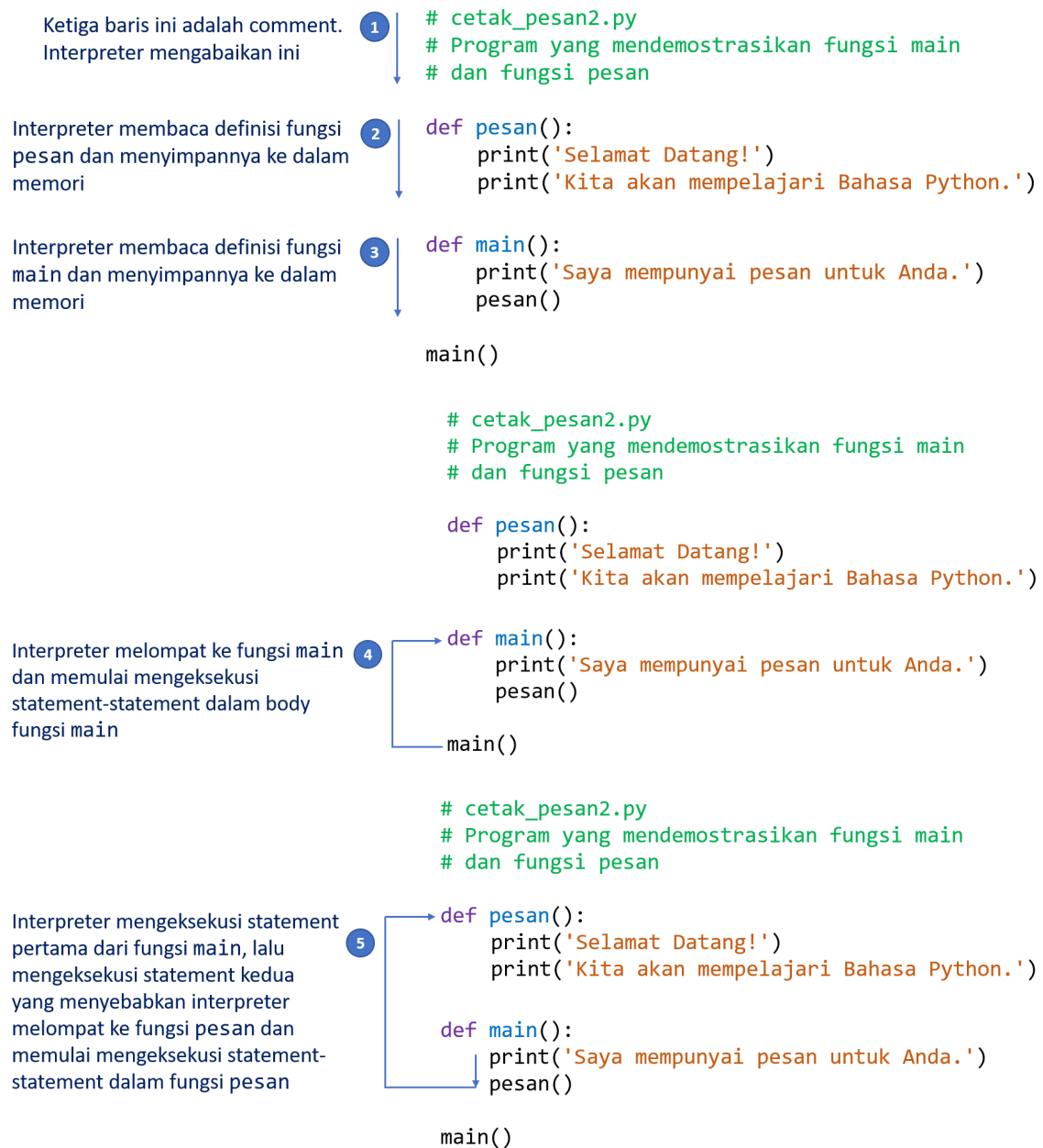
Output dari program `cetak_pesan2.py` di atas:

```
Saya mempunyai pesan untuk Anda.  
Selamat Datang!  
Kita akan mempelajari Bahasa Python.
```

Gambar berikut menjelaskan bagian-bagian dari program `cetak_pesan2.py`:



Gambar berikut menjelaskan alur eksekusi program cetak_pesanan2.py :



Interpreter mengeksekusi statement-statement dalam fungsi pesan. Lalu setelah semua statement selesai dieksekusi, interpreter melompat ke bagian program yang memanggil fungsi pesan, dan melanjutkan eksekusi pada bagian program tersebut

```
# cetak_pesanan2.py
# Program yang mendemostrasikan fungsi main
# dan fungsi pesan

def pesan():
    print('Selamat Datang!')
    print('Kita akan mempelajari Bahasa Python.')

def main():
    print('Saya mempunyai pesan untuk Anda.')
    pesan()

main()
```

Karena tidak ada lagi statement yang dieksekusi di dalam fungsi main, interpreter melompat ke bagian program yang memanggil fungsi main

```
# cetak_pesanan2.py
# Program yang mendemostrasikan fungsi main
# dan fungsi pesan

def pesan():
    print('Selamat Datang!')
    print('Kita akan mempelajari Bahasa Python.')

def main():
    print('Saya mempunyai pesan untuk Anda.')
    pesan()

main()
```

Karena tidak ada lagi statement yang dieksekusi, interpreter mengakhiri program

```
# cetak_pesanan2.py
# Program yang mendemostrasikan fungsi main
# dan fungsi pesan

def pesan():
    print('Selamat Datang!')
    print('Kita akan mempelajari Bahasa Python.')

def main():
    print('Saya mempunyai pesan untuk Anda.')
    pesan()

main()
Keluar dari program
```

2.5.3 Fungsi yang Menerima Argumen

Sebuah fungsi dapat menerima data ketika dipanggil. Data yang diterima oleh fungsi ini disebut sebagai **argumen**. Bentuk dasar syntax penulisan definisi fungsi yang menerima argumen:

```
def <nama_fungsi>(<parameter>, <parameter>, ...):
    <statement>
    <statement>
    ...
    <statement>
```

Parameter adalah variabel yang digunakan untuk menyimpan nilai argumen

Pada penulisan definisi fungsi yang menerima argumen, kita menuliskan daftar parameter di dalam tanda kurung setelah nama fungsi. **Parameter** adalah variabel yang digunakan untuk menyimpan nilai argumen yang diberikan, sehingga argumen tersebut dapat kita gunakan di dalam body fungsi.


Berikut adalah contoh definisi fungsi yang menerima sebuah argumen:

```
def sapa(tamu):  
    print(f'Selamat datang, {tamu}')
```

Pada definisi fungsi `sapa` di atas kita mendefinisikan parameter `tamu` untuk menyimpan nilai argumen yang diberikan. Gambar berikut mengilustrasikan ini:

```
def sapa(tamu):  
    print(f'Selamat datang, {tamu}')
```

Parameter bernama `tamu` digunakan untuk menyimpan argumen



Untuk memanggil fungsi yang menerima argumen, kita menuliskan nilai argumen di dalam tanda kurung setelah nama fungsi. Sebagai contoh, kita dapat memanggil fungsi `sapa` dengan statement seperti berikut:

```
sapa('Budi')
```

Statement di atas akan memberikan output:

```
Selamat datang, Budi
```

Ketika statement pemanggilan fungsi `sapa` di atas dieksekusi, parameter `tamu` dibuat dalam memori, lalu argumen string `'Budi'` disalin ke parameter `tamu` tersebut, kemudian statement-statement di dalam body fungsi `sapa` dieksekusi. Setelah selesai mengeksekusi semua statement dalam fungsi `sapa`, parameter `tamu` dihapus dari memori. Gambar berikut mengilustrasikan proses pemanggilan fungsi `sapa`:

Pemanggilan Fungsi
Memberikan argumen

Definisi Fungsi
Argumen disimpan dalam parameter

```
sapa('Budi')
```

```
def sapa(tamu):  
    print(f'Selamat datang, {tamu}!')
```

tamu



Kita dapat juga menuliskan definisi fungsi yang menerima lebih dari satu argumen. Program berikut mendefinisikan sebuah fungsi `cetak_jumlah` yang menerima dua argumen bertipe numerik dan menampilkan jumlah kedua argumen tersebut:


```
# multi_argument.py
# Program ini mendemonstrasikan sebuah fungsi yang menerima dua argumen
def cetak_jumlah(num1, num2):
    result = num1 + num2
    print(result)

def main():
    print('Jumlah dari 12 dan 45 adalah')
    cetak_jumlah(12, 45)

main()
```

Output dari program di atas:

```
Jumlah dari 12 dan 45 adalah
57
```

2.5.4 Fungsi yang Mengembalikan Nilai

Pada pembahasan fungsi built-in kita telah melihat bahwa fungsi umumnya mengembalikan suatu nilai. Untuk membuat sebuah fungsi yang mengembalikan nilai, di dalam body kita harus menuliskan statement `return`.

Syntax umum penulisan definisi fungsi yang mengembalikan nilai adalah sebagai berikut:

```
def <nama_fungsi>(<parameter>, <parameter>, ...):
    <statement>
    <statement>
    ...
    return <ekspresi>
```

Statement return
 Nilai ekspresi setelah keyword return akan dikembalikan ke pemanggil fungsi

Berikut adalah contoh fungsi yang mengembalikan nilai:

```
def sum(num1, num2):
    result = num1 + num2
    return result
```

Fungsi `sum` di atas menerima dua buah argumen, menjumlahkan kedua argumen tersebut, lalu mengembalikan hasilnya ke pemanggil fungsi.

Umumnya ketika kita memanggil fungsi yang mengembalikan nilai, kita ingin mengolah lebih lanjut nilai yang dikembalikan tersebut. Sehingga, umumnya kita menugaskan nilai kembali tersebut ke suatu variabel. Sebagai contoh, berikut adalah contoh statement yang memanggil fungsi `sum` dan menyimpan nilai kembali dari pemanggilan fungsi tersebut ke variabel bernama `total`:

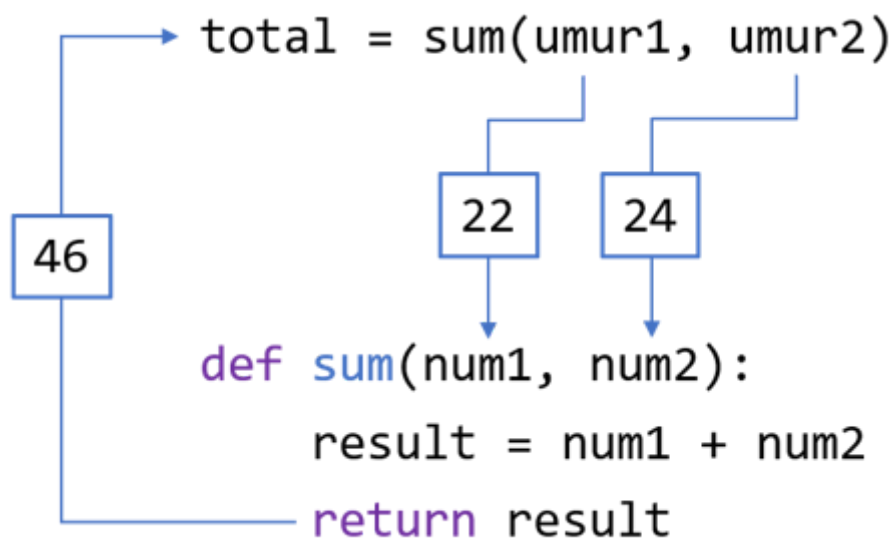
```
total = sum(22, 24)
```

Statement di atas memanggil fungsi `sum` dengan memberikan argumen `22` dan `24` lalu menyimpan nilai kembali dari pemanggilan fungsi tersebut ke variabel `total`. Setelah statement di atas dieksekusi, variabel `total` akan menyimpan nilai `46`.

Kita juga dapat menuliskan nama variabel sebagai argumen ke pemanggilan fungsi. Sebagai contoh:

```
umur1 = 22
umur2 = 24
total = sum(umur1, umur2)
```

Ketika statement terakhir: `total = sum(umur1, umur2)` dieksekusi, nilai yang disimpan oleh variabel `umur1` disalin ke parameter `num1` dan nilai yang disimpan oleh variabel `umur2` disalin ke parameter `num2`. Lalu, nilai kedua parameter tersebut dijumlahkan dan hasilnya ditugaskan ke variabel `total`. Gambar berikut mengilustrasikan proses pengekseskuan statement `total = sum(umur1, umur2)` pada kode di atas:



Contoh program yang memanggil fungsi `sum()`:

```
# total_umur.py
# Program ini menggunakan nilai kembali dari sebuah fungsi.

# Fungsi sum menerima dua argumen numerik dan
# mengembalikan jumlah dari kedua argument
def sum(num1, num2):
    result = num1 + num2
    return result

# Fungsi main
def main():
    umur1 = int(input('Masukkan umur Anda: '))
    umur2 = int(input('Masukkan umur teman baik Anda: '))
    # Jumlahkan keduanya dengan memanggil fungsi sum
    total = sum(umur1, umur2)
    print('Umur Anda dan teman Anda: ', total, 'tahun.')

# Panggil fungsi main
main()
```

Output dari program `total_umur.py` di atas:

```
Masukkan umur Anda: 22
Masukkan umur teman baik Anda: 24
UmurAnda dan teman Anda: 46 tahun.
```

Pada statement `return`, kita tidak hanya dapat menuliskan variabel setelah keyword `return`, namun kita juga dapat menuliskan sebuah ekspresi. Sebagai contoh, fungsi `sum` sebelumnya dapat kita tulis ulang menjadi seperti berikut:

```
# Versi 2 definisi fungsi sum
def sum(num1, num2):
    return num1 + num2
```

Perhatikan pada definisi fungsi di atas kita menuliskan langsung ekspresi yang menjumlahkan parameter `num1` dan `num2` pada statement `return`. Penulisan ekspresi pada statement `return` ini lebih efisien karena hasil perhitungan tidak perlu disimpan ke sebuah variabel terlebih dahulu.

Berikut adalah contoh lain fungsi yang mengembalikan nilai dengan penulisan ekspresi pada statement `return`:

```
def luas_lingkaran(radius):
    pi = 3.14159265358979
    return pi * radius * radius
```

Berikut adalah contoh program yang menggunakan fungsi `luas_lingkaran`:

```
# hitung_luas_lingkaran.py
# Program ini menghitung luas lingkaran
# dari radius yang diberikan pengguna
def luas_lingkaran(radius):
    pi = 3.14159265358979
    return pi * radius * radius

def main():
    rad = float(input('Masukkan radius lingkaran (cm): '))
    luas = luas_lingkaran(rad)
    print(f'Luas lingkaran = {luas:.2f} cm2')

main()
```

Contoh output dari program `hitung_luas_lingkaran.py` di atas:

```
Masukkan radius lingkaran (cm): 11
Luas lingkaran = 380.13 cm2
```

Sejauh ini kita hanya melihat contoh-contoh fungsi yang mengembalikan nilai numerik. Kita dapat menuliskan fungsi untuk mengembalikan nilai tipe apapun, termasuk nilai string:

```
def ambil_nama():
    # Minta nama dari pengguna
    nama = input('Masukkan nama Anda: ')
    # Kembalikan nama
    return nama
```

Fungsi `ambil_nama` di atas meminta pengguna memasukkan sebuah string lalu mengembalikan string yang dimasukkan pengguna tersebut ke pemanggil fungsi.

2.5.5 Variabel Lokal

Variabel-variabel yang kita tuliskan dalam definisi fungsi disebut dengan **variabel lokal**. Kata lokal dalam variabel lokal berarti variable ini hanya dapat digunakan secara lokal di dalam fungsi dimana variabel tersebut didefinisikan. Interpreter membuat variabel lokal hanya ketika fungsi dipanggil dan menghapusnya ketika fungsi selesai dijalankan.

Perhatikan definisi fungsi `luas_lingkaran` berikut:

```
def luas_lingkaran(radius):
    pi = 3.14159265358979
    return pi * radius * radius
```

Variabel `pi` yang didefinisikan pada baris kedua adalah variabel lokal dari fungsi `luas_lingkaran`. Variabel `pi` ini dibuat dalam memori oleh interpreter hanya ketika fungsi `luas_lingkaran` dipanggil dan setelah selesai mengeksekusi statement-statement dalam fungsi `luas_lingkaran`, variabel `pi` tersebut dihapus dalam memori. Ini berarti, variabel `pi` hanya dapat diakses (digunakan) oleh statement-statement yang berada di dalam body fungsi `luas_lingkaran`.

Program di bawah memperlihatkan error ketika kita mencoba mengakses variabel lokal di luar fungsi dimana variabel tersebut didefinisikan:

```
# luas_lingkaran_error.py
# Program ini menghasilkan ERROR

def luas_lingkaran(radius):
    pi = 3.14159265358979
    return pi * radius * radius

def main():
    luas = luas_lingkaran(15)
    print(luas)
    print(pi) # ERROR! pi hanya dapat diakses dari fungsi luas_lingkaran

main()
```

Ketika kita mencoba menjalankan program di atas, kita akan mendapatkan pesan error seperti berikut:

```
Traceback (most recent call last):
  File "luas_lingkaran_error.py", line 12, in <module>
    main()
  File "luas_lingkaran_error.py", line 10, in main
    print(pi) # ERROR!! pi hanya dapat diakses dari fungsi luas_lingkaran
NameError: name 'pi' is not defined
```

Pesan error ini mengatakan bahwa variabel `pi` tidak didefinisikan di dalam fungsi `main`. Hal ini terjadi karena di dalam fungsi `main` pada statement terakhir: `print(pi)`, kita mencoba mengakses variabel `pi` yang merupakan variabel lokal pada fungsi `luas_lingkaran`.

2.5.6 Variabel Konstan Global

Jika kita menuliskan sebuah variabel di luar semua definisi fungsi, variabel tersebut dapat diakses oleh statement-statement dimanapun dalam program, termasuk oleh statement-statement di dalam definisi fungsi. Variabel yang didefinisikan di luar semua definisi fungsi ini disebut dengan **variabel global** (global berarti dapat diakses dari manapun dalam program).

Perhatikan program berikut:

```
# global_variabel.py

pi = 3.14159265358979

def luas_lingkaran(radius):
    return pi * radius * radius

def main():
    luas = luas_lingkaran(15)
    print(luas)
    print(pi)

main()
```

Variabel `pi` yang didefinisikan pada baris 3 adalah variabel global. Variabel `pi` ini dapat diakses di bagian manapun dalam program termasuk dalam body fungsi `luas_lingkaran` maupun dalam body fungsi `main`.

```
pi = 3.14159265358979
def luas_lingkaran(radius):
    return pi * radius * radius
def main():
    luas = luas_lingkaran(15)
    print(luas)
    print(pi)
main()
```

Variabel `pi` didefinisikan di luar semua definisi fungsi. Variabel ini bisa diakses dari bagian manapun dalam program

Variabel `pi` dapat digunakan di dalam fungsi `luas_lingkaran` dan juga di dalam fungsi `main`

Dalam praktik pemrograman yang baik, kita harus menghindari penggunaan variabel global, karena akan sulit melacak perubahan-perubahan nilai yang terjadi pada variabel global ini. Umumnya kita memanfaatkan variabel global ini sebagai **konstan global** (variabel yang tidak dapat diubah nilainya dan dapat diakses secara global). Python tidak mempunyai fitur untuk membuat variabel konstan, sehingga umumnya programmer mengikuti konvensi dengan

menggunakan huruf besar dalam penamaan konstan global dan tidak mengubah nilainya di dalam program. Sebagai contoh, dalam praktik pemrograman yang baik, variabel `pi` pada program `global_variabel.py` sebaiknya dinamakan dengan `PI` untuk menandakan bahwa variabel tersebut adalah konstan global:

```
PI = 3.14159265358979
```

Variabel global dinamakan dengan huruf besar untuk menandakan ini adalah konstan global.

```
def luas_lingkaran(radius):  
    return PI * radius * radius
```

Berikut adalah program yang menggunakan konstan global:

```
# lingkaran.py  
# Program ini menghitung luas lingkaran dan  
# keliling lingkaran dari radius yang diberikan pengguna  
  
PI = 3.14159265358979    # Konstan Global  
  
def luas_lingkaran(radius):  
    return PI * radius * radius    # menggunakan konstan global PI  
  
def keliling_lingkaran(radius):  
    return 2 * PI * radius    # menggunakan konstan global PI  
  
def main():  
    radius = float(input('Masukkan radius lingkaran (cm): '))  
    luas = luas_lingkaran(radius)  
    keliling = keliling_lingkaran(radius)  
    print(f'Luas lingkaran dengan radius {radius:.2f} = {luas:.2f} cm2')  
    print(f'keliling lingkaran dengan radius {radius:.2f} = {keliling:.2f} cm')  
  
main()
```

Berikut adalah contoh output program `lingkaran.py` di atas:

```
Masukkan radius lingkaran (cm): 10  
Luas lingkaran dengan radius 10.00 = 314.16 cm2  
keliling lingkaran dengan radius 10.00 = 62.83 cm
```

2.6 Standard Library

Python menyertakan standard library yang berisi kumpulan module-module. Module adalah kumpulan fungsi-fungsi dan variable-variable konstan. Terdapat dua module dalam standard library yang sering digunakan: module `math` dan module `random`. Kita akan membahas kedua module ini.

2.6.1 Module `math`

Module `math` berisi fungsi-fungsi matematika dan variabel-variabel matematika. Untuk menggunakan suatu module, kita pertama kali harus mengimpornya dengan menuliskan statement `import`:

```
import <nama_module>
```

Misalkan untuk mengimport module `math` kita menuliskan statement `import` berikut:

```
import math
```

Untuk menggunakan suatu fungsi dalam suatu module, kita menuliskan statement pemanggilan fungsi dengan notasi dot:

```
<nama_module>.<nama_fungsi>()
```

Perhatikan tanda titik di antara `<nama_module>` dan `<nama_fungsi>`.

Sebagai contoh, di dalam module `math` terdapat fungsi dengan nama `sqrt` yang menghitung akar kuadrat dari argumen. Kita memanggil fungsi `sqrt` tersebut menggunakan notasi dot seperti berikut:

```
math.sqrt(<argumen>)
```

Dan misalkan kita ingin menghitung akar kuadrat dari 64, maka kita dapat menggunakan fungsi `sqrt` seperti berikut:

```
math.sqrt(45)
```

Kita dapat melihat fungsi-fungsi dan variabel-variabel apa saja yang terdapat dalam suatu module dengan menggunakan fungsi `help()` pada modus interaktif:

```
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

[Masih banyak fungsi-fungsi lain yang tidak ditampilkan disini.]
```

Program berikut mendemonstrasikan penggunaan module `math`:

```
# akar_kuadrat.py
# Program ini mendemonstrasikan fungsi sqrt dalam module math

import math    # Import module math

def main():
    # Minta angka dari pengguna
```

```

num = float(input('Masukkan sebuah angka: '))
# Hitung akar kuadrat dari num menggunakan
# fungsi sqrt dalam module math
akar = math.sqrt(num)
# Tampilkan akar dari num
print(f'Akar kuadrat dari {num} adalah {akar}.')

# Panggil fungsi main
main()

```

Berikut adalah contoh output dari program `akar_kuadrat.py` di atas:

```

Masukkan sebuah angka: 9
Akar kuadrat dari 9.0 adalah 3.0.

```

Selain fungsi, di dalam module `math` juga terdapat variabel-variabel konstan yang dapat digunakan (seperti `pi`). Program berikut mendemonstrasikan penggunaan variabel konstan `pi` dalam module `math`:

```

# lingkaran2.py
# Program ini menghitung luas dan kel

import math

def luas_lingkaran(radius):
    return math.pi * radius * radius

def keliling_lingkaran(radius):
    return 2 * math.pi * radius

def main():
    radius = float(input('Masukkan radius lingkaran (cm): '))
    luas = luas_lingkaran(radius)
    keliling = keliling_lingkaran(radius)
    print(f'Luas lingkaran dengan radius {radius:.2f} = {luas:.2f} cm2')
    print(f'keliling lingkaran dengan radius {radius:.2f} = {keliling:.2f} cm')

main()

```

Berikut adalah contoh output dari program `lingkaran2.py` di atas:

```

Masukkan radius lingkaran (cm): 9
Luas lingkaran dengan radius 9.00: 254.47 cm2
keliling lingkaran dengan radius 9.00: 56.55 cm

```

Tabel berikut mendaftar fungsi-fungsi dan variabel-variabel yang sering digunakan dalam module `math`:

Fungsi/Konstanta	Matematika	Keterangan
<code>math.pi</code>	π	Nilai aproksimasi dari pi.
<code>math.e</code>	e	Nilai aproksimasi dari e .
<code>math.sqrt(x)</code>	\sqrt{x}	Akar kuadrat dari x .
<code>math.sin(x)</code>	$\sin x$	Sinus dari x .
<code>math.cos(x)</code>	$\cos x$	Cosinus dari x .
<code>math.tan(x)</code>	$\tan x$	Tangen dari x .
<code>math.asin(x)</code>	$\arcsin x$	Inverse sinus dari x .
<code>math.acos(x)</code>	$\arccos x$	Inverse cosinus dari x .
<code>math.atan(x)</code>	$\arctan x$	Inverse tangen dari x .
<code>math.log(x)</code>	$\ln x$	Logaritma natural (basis e) dari x .
<code>math.log10(x)</code>	$\log x$	Logaritma basis 10 dari x .
<code>math.exp(x)</code>	e^x	Eksponensial dari x .
<code>math.ceil(x)</code>	$\lceil x \rceil$	Angka bulat terkecil $\geq x$.
<code>math.floor(x)</code>	$\lfloor x \rfloor$	Angka bulat terbesar $\leq x$.

2.6.2 Module `random`

Module `random` berisi fungsi-fungsi yang dapat digunakan untuk menghasilkan angka acak. Salah satu fungsi dalam module `random` adalah fungsi `randint` yang menghasilkan sebuah integer acak dalam suatu interval. Pemanggilan fungsi `randint` dituliskan dengan format seperti berikut:

```
random.randint(<bawah>, <atas>)
```

Pemanggilan fungsi `randint` akan mengembalikan integer acak di antara nilai `<bawah>` dan nilai `<atas>`. Sebagai contoh,

```
random.randint(0, 10)
```

menghasilkan integer antara 0 s.d 10 (termasuk 0 dan 10). Sehingga, nilai integer yang dikembalikan dapat berupa nilai 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, atau 10.

Program berikut mencontohkan penggunaan fungsi `randint` dalam module `random`:

```
# angka_acak.py
# Program ini menampilkan sebuah integer acak
# dalam interval 0 s.d 10.

import random

def main():
    print('Integer acak: ', random.randint(0, 10))

main()
```

Contoh output dari program `angka_acak.py` di atas:

```
Integer acak: 7
```

Selain fungsi `randint`, terdapat fungsi-fungsi lain dalam module `random`, dua diantaranya adalah:

- `random.random()`: mengembalikan floating point acak antara 0 dan 1.
- `random.randrange()`: dapat dipanggil dengan satu, dua, atau tiga argumen; mengembalikan integer acak dalam interval.

Sesi interaktif berikut mendemonstrasikan `random.random()`:

```
>>> import random
>>> random.random()
0.8761336035184462
>>> random.random()
0.7698035311730363
>>> random.random()
0.387495752677397
```

Sesi interaktif berikut mendemonstrasikan `random.randrange()`:

```
>>> import random
>>> random.random()
0.8761336035184462
>>> random.random()
0.387495752677397
>>> random.randrange(0, 100)
97
>>> random.randrange(0, 100)
24
>>> random.randrange(0, 101, 10)
90
>>> random.randrange(0, 101, 10)
70
```

Satu argumen: `random.randrange(x)`
mengembalikan integer acak antara 0 s.d $x-1$

Dua argumen: `random.randrange(x, y)`
mengembalikan integer acak antara x s.d $y-1$

Tiga argumen: `random.randrange(x, y, a)`
mengembalikan salah satu dari barisan integer
 $[x, x+a, x+2a, \dots, a-1]$.

Contoh: `random.randrange(0, 101, 10)`
mengembalikan salah satu dari barisan integer
 $[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$

REFERENSI

[1] Gaddis, Tony. 2012. Starting Out With Python Second Edition. United States of America: Addison-Wesley.