

# PEMROGRAMAN PYTHON (1)

Buku jilid 1 ini membahas tentang teknik pemrograman dasar terstruktur  
menggunakan bahasa pemrograman Python versi 3.X

*Disusun oleh:  
Rosihan Ari Yuana,  
S.Si, M.Kom*

## Daftar Isi

Daftar Isi.....	1
Pendahuluan .....	3
Instalasi Python .....	3
Setting Path Manual .....	4
Mengenal Python IDLE ( <i>Python Integrated DeveLopment Environment</i> ).....	6
Mengenal Ekspresi .....	6
Operator .....	7
Tipe Data.....	8
Operator Khusus Untuk String .....	8
Operator String Concatenation (Penggabungan String) .....	8
Operator String Replication (Pengulangan String) .....	8
Menyimpan Nilai ke Variabel .....	9
Aturan Pemberian Nama Variabel.....	9
Sifat Case Sensitivitas Variabel.....	10
Membuat Program dengan Python.....	10
Statement Kontrol.....	14
Tipe Data Boolean .....	14
Operator Tipe Data Boolean .....	15
Operator Relasional.....	15
Statement Kondisional - IF .....	16
Statement Perulangan – WHILE .....	21
Infinite Loop .....	27
Statement Perulangan – FOR .....	27
Ekuivalensi Perulangan WHILE dan FOR .....	31
Soal-soal Latihan.....	32
Functions .....	35
Cara Mendefinisikan Function .....	36
Function dengan <i>Return Value</i> .....	38
Lebih Lanjut dengan Function <code>print ( )</code> .....	39

Mengimport Function dari File Lain .....	40
Variabel Lokal dan Global .....	41
Penanganan Exception ( <i>Exception Handling</i> ) .....	43

## Pendahuluan

Python merupakan bahasa pemrograman tingkat tinggi (high level language) yang bersifat multi-purpose, dan sangat mudah untuk dipelajari. Seperti halnya PHP dan PERL, Python termasuk jenis bahasa program yang dijalankan melalui interpreter, artinya ketika akan menjalankannya tidak perlu melakukan kompilasi terlebih dahulu, tidak seperti Pascal, Java, C/C++ dsb.

Python dikembangkan awalnya oleh Guido van Rossum selama tahun 1985 s/d 1990, hingga saat ini masih terus dikembangkan. Saat ini terdapat 2 jenis versi Python, yaitu versi 2.x dan 3.x. Adapun versi 2.x yang terakhir dirilis adalah versi 2.7 (di pertengahan tahun 2010). Sedangkan yang versi 3.x mulai dirilis tahun 2008. Perbaikan yang paling drastis di versi 3.x dibandingkan 2.x adalah dukungan Unicode yang lebih baik (dengan menjadikan string teks menjadi Unicode secara default). Selain itu, beberapa aspek perintah utama (seperti `print` dan `exec` menjadi sebuah fungsi) telah disesuaikan agar lebih mudah bagi programmer pemula untuk belajar dan lebih konsisten dengan bahasa lainnya. Dari sisi performa, versi 3.x memiliki performa yang lebih baik dibandingkan 2.x.

Adapun kelebihan Python dibandingkan bahasa pemrograman lainnya adalah:

1. Mendukung OOP (object oriented programming)
2. Multi platform, Python dapat diinstall di beberapa OS (Windows, Unix/Linux, Mac OSX, IBM i, iOS, Solaris)
3. Python bersifat opensource
4. Struktur dan perintah Python mudah untuk dipahami bagi programmer pemula
5. Banyak tersedia module/library yang siap pakai untuk keperluan tertentu
6. Python bisa diintegrasikan dengan beberapa DBMS yang populer (MySQL, Oracle dll)
7. Script Python bersifat portable
8. Mendukung pemrograman berbasis GUI (Graphical User Interface)
9. Python dapat diintegrasikan dengan program yang dibuat dengan bahasa lain spt C/C++, Java
10. Python mendukung pemrograman socket atau COM
11. Meskipun berbasis interpreter, kode program Python juga dapat dijadikan byte-code untuk aplikasi skala besar melalui proses kompilasi

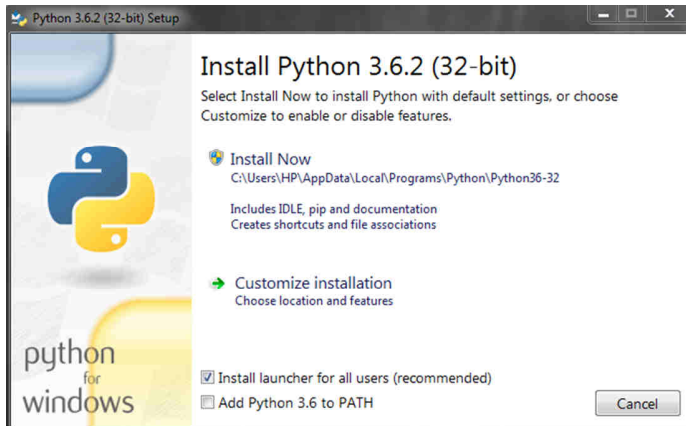
Dalam tutorial ini, pembahasan hanya difokuskan pada penggunaan Python 3.x saja.

## Instalasi Python

Untuk bisa menjalankan program yang dibuat dengan bahasa Python, dibutuhkan Python interpreter yang bisa diunduh di <https://www.python.org/downloads/>

Berikut ini langkah-langkah untuk instalasi Python intrepeternya:

1. Jalankan installer Pythonnya
2. Selanjutnya akan muncul kotak dialog



Keterangan:

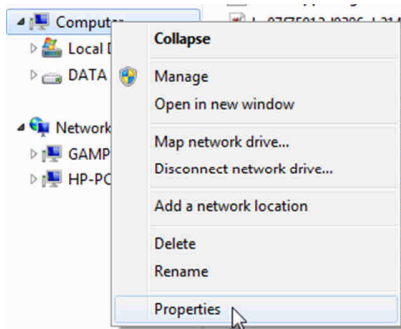
- Apabila option 'Install Now' diklik maka instalasi akan langsung dijalankan, dan Python akan diinstall di direktori default yang ditentukan
  - Apabila option 'Customize Instalation' diklik maka kita diminta menentukan lokasi direktori secara manual dan fitur apa saja yang diinginkan untuk diinstal
  - Apabila checkbox 'Add Python to PATH' dipilih, maka installer sekaligus akan mensetting Python path direktori ke dalam path Windowsnya.
3. Lanjutkan hingga proses instalasi selesai

## Setting Path Manual

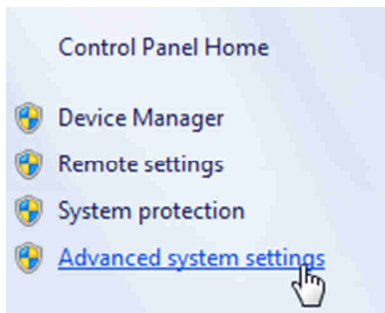
Setelah Python interpreternya diinstall, langkah selanjutnya adalah melakukan setting path. Langkah ini dilakukan supaya sistem operasi bisa menjalankan program/script Python di direktori mana saja. Namun apabila pada tahap instalasi yang telah dibahas sebelumnya kita sudah memilih option 'Add Python to PATH' maka langkah setting path secara manual ini tidak diperlukan.

Berikut ini adalah cara melakukan setting path di OS Windows secara manual:

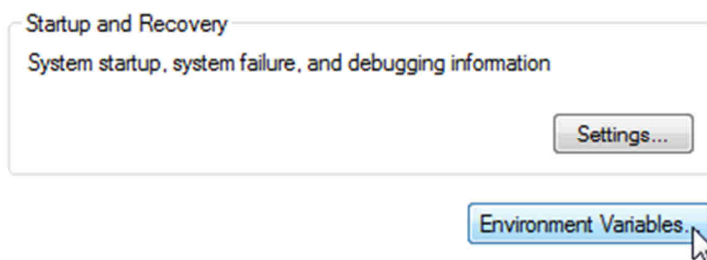
1. Klik kanan pada My Computer



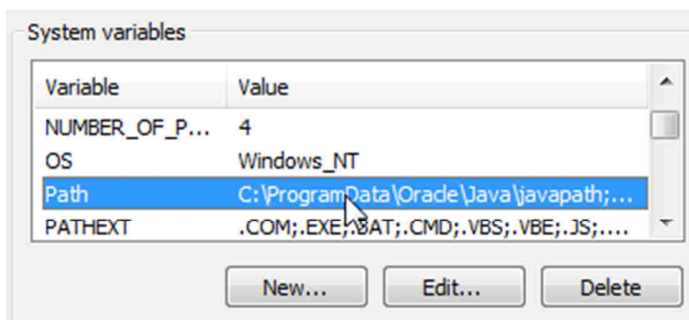
2. Pilih Properties
3. Pilih Advanced System Settings



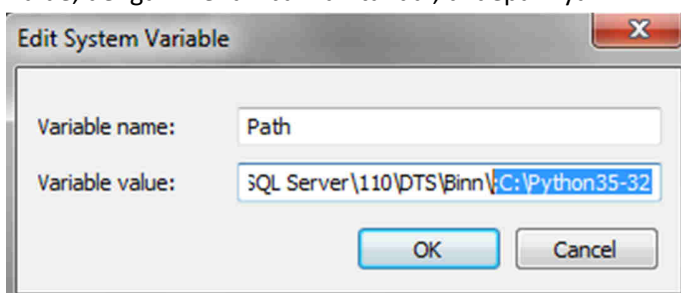
4. Pilih Environment Variables



5. Pilih Path



6. Klik tombol Edit
7. Tambahkan path menuju ke direktori tempat Python diinstall pada akhir baris dalam Variable Value, dengan menambahkan tanda ; di depannya



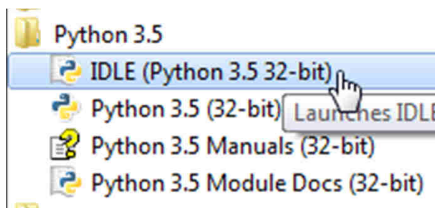
8. Kemudian restart komputer

## Mengenal Python IDLE (*Python Integrated DeveLopment Environment*)

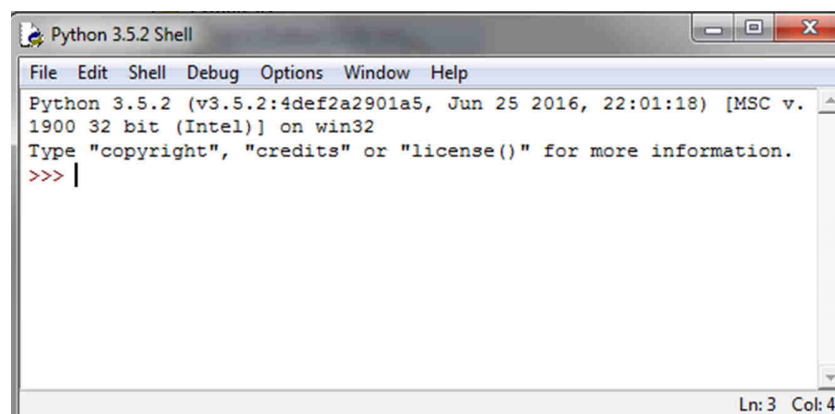
Python IDLE merupakan paket fasilitas yang diberikan kepada programmer Python supaya lebih mudah dalam membuat program dan menjalankannya. Tanpa menggunakan Python IDLE, kita agak sedikit lebih repot ketika membuat kode program, kemudian menjalankannya sebagaimana telah dijelaskan sebelumnya. Dengan menggunakan Python IDLE, kode program ditulis, menjalankannya, dan tampilan outputnya hanya melalui sebuah aplikasi saja, tanpa ribet.

Untuk membuka Python IDLE, caranya adalah:

1. Klik Start – Programs pada Windows
2. Pilih menu IDLE



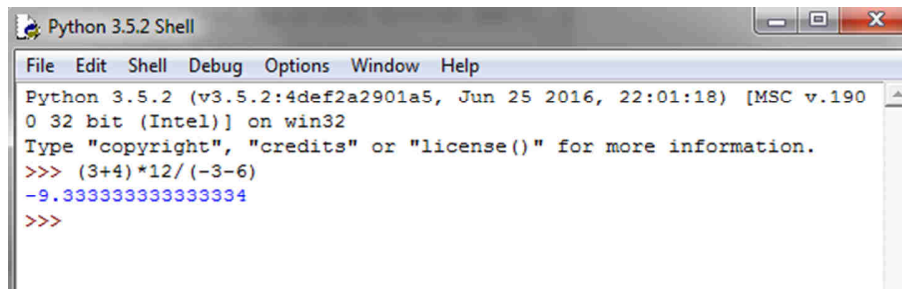
Selanjutnya akan muncul tampilan Python Shell sebagai berikut



Di dalam Python Shell inilah, kita bisa menuliskan berbagai perintah ekspresi yang diperlukan untuk proses komputasi.

## Mengenal Ekspresi

Di dalam Python, kita bisa menuliskan sebuah ekspresi melalui Python Shell. Ekspresi adalah suatu instruksi atau perintah yang diberikan dari programmer kepada komputer untuk dilakukan. Berikut ini contoh ekspresi aritmatika untuk menghitung nilai dari  $(3+4) \times 12 / (-3-6)$  secara langsung melalui Python Shell.



```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.190
0 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> (3+4)*12/(-3-6)
-9.333333333333334
>>>

```

Dari tampilan yang dihasilkan, akan muncul output hasil perhitungannya yaitu -9.333333333333334.

## Operator

Dalam sebuah ekspresi, bisa terdapat satu atau lebih operator. Pada contoh ekspresi sebelumnya, yaitu  $(3+4)*12/(-3-6)$  terdapat 4 operator aritmatika, yaitu + (penjumlahan), \* (perkalian), / (pembagian), dan – (pengurangan).

Secara umum, dalam Python dikenal beberapa operator aritmatika, yaitu:

Simbol Operator	Operasi	Contoh	Hasil	Level (presedensi)
**	Perpangkatan	2**3	8	1
%	Modulus (sisa hasil bagi)	7%2	1	2
//	Pembagian yang menghasilkan bilangan bulat (pembulatan ke bawah)	10//4	2	
/	Pembagian yang menghasilkan bilangan riil	10/4	2.5	
*	Perkalian	2*7	14	
-	Pengurangan	2-7	-5	3
+	Penjumlahan	2+7	9	

Dari tabel di atas, operator \*\* memiliki level 1, artinya operator ini mendapat prioritas pertama (tingkat presedensi) untuk diselesaikan terlebih dahulu. Selanjutnya operator %, //, /, dan \* sama-sama memiliki level ke-2 yang akan diselesaikan berikutnya. Sedangkan operator – dan + memiliki level paling rendah, yang akan dijalankan setelah level ke-2. Sehingga dalam hal ini, apabila diberikan ekspresi sebagai berikut:

$2**3*3-12/2+4$

Akan dihasilkan nilai 22, karena ekspresi tersebut ekuivalen dengan  $((2**3)*3)-(12/2)+4$ .



## Tipe Data

Dalam pemrograman komputer, setiap nilai yang dihasilkan dari sebuah ekspresi memiliki tipe data tertentu. Tipe data ini menentukan jenis atau kategori dari suatu nilai. Seperti halnya di dalam matematika, bilangan terdapat beberapa jenis yaitu misalnya bilangan bulat (... , -3, -2, -1, 0, 1, 2, ...) atau bilangan riil (-3.5, -5.123, 0.234, 3,123, dll).

Di dalam Python, terdapat 3 tipe data utama yaitu:

Tipe Data	Contoh
Integer (bil bulat)	-2, -1, 0, 1, 2
Floating Point (bil riil)	-1.212, 0.341, 5.234
String	'a', 'aa', 'python', 'rosihan ari', '(0271) 783243', 'K3517001'

## Operator Khusus Untuk String

Dalam bab sebelumnya tentang operator telah disebutkan beberapa operator aritmatika yang hanya berlaku bagi nilai bertipe data angka (integer dan floating point). Sedangkan pada bab ini, akan diberikan beberapa operator yang berlaku untuk string.

### Operator String Concatenation (Penggabungan String)

Untuk menggabungkan 2 string atau lebih dalam Python menggunakan operator +.

```
'Hallo' + 'Selamat Pagi?'
'HalloSelamat Pagi?'
```

```
'Assalaamu\'alaikum'
"Assalaamu'alaikum"
```

```
'Assalaamu\'alaikum...' + 'Selamat' + 'Pagi?'
"Assalaamu'alaikum...SelamatPagi?"
```

### Operator String Replication (Pengulangan String)

Sebuah string bisa ditulis berulang sampai dengan n kali, dengan menggunakan operator \*.

```
'Hello' * 3
'HelloHelloHello'
```

```
'Hello' * (2+3)
'HelloHelloHelloHelloHello'
```

## Menyimpan Nilai ke Variabel

Sebuah nilai dapat disimpan ke dalam sebuah variabel. Fungsi variabel dalam pemrograman komputer dapat diibaratkan seperti kotak penyimpanan. Nilai yang tersimpan dalam sebuah variabel dapat diambil kembali guna dioperasikan lagi untuk ekspresi lainnya.

Untuk menyimpan sebuah nilai ke dalam variabel, caranya adalah menggunakan operator assignment (=).

```
bil1 = 10
bil2 = 20
hasil = bil1 + bil2
hasil
30
```

```
kata1 = 'Hello'
kata2 = kata1 + 'World'
kata2
'HelloWorld'
```

```
x = 1          # memberi nilai 1 kepada variabel x
x = x + 1      # variabel x diberi nilai 2 dari hasil 1 + 1
x = x + 1      # variabel x diberi nilai 3 dari hasil 2 + 1
x              # mencetak nilai x terakhir, yaitu 3
3
```

## Aturan Pemberian Nama Variabel

Pemberian nama sebuah variabel pada prinsipnya bisa sembarang, asal memenuhi ketentuan sbb:

1. Tidak boleh diawali dengan angka
2. Tidak boleh dipisahkan dengan spasi
3. Hanya boleh terdiri dari angka, huruf, dan underscore

Meskipun pemberian nama variabel adalah sembarang, namun sangat disarankan sesuai dengan maksud nilai yang akan disimpan di dalamnya. Perhatikan contoh rangkaian ekspresi berikut ini untuk menghitung luas segitiga dengan panjang alas 3 dan tinggi 7 satuan luas.

```
alasSegitiga = 3
tinggiSegitiga = 7
luasSegitiga = alasSegitiga * tinggiSegitiga * 0.5
luasSegitiga
10.5
```

## Sifat Case Sensitivitas Variabel

Di dalam Python, besar kecilnya huruf dalam penulisan nama variabel akan berpengaruh (case sensitive). Sebagai contoh misalkan variabel `alasSegitiga` dengan `AlasSegitiga` adalah dua variabel yang berbeda. Perhatikan contoh berikut ini

```

alasSegitiga = 10
tinggiSegitiga = 5
luasSegitiga = AlasSegitiga * tinggiSegitiga * 0.5
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    luasSegitiga = AlasSegitiga * tinggiSegitiga * 0.5
NameError: name 'AlasSegitiga' is not defined

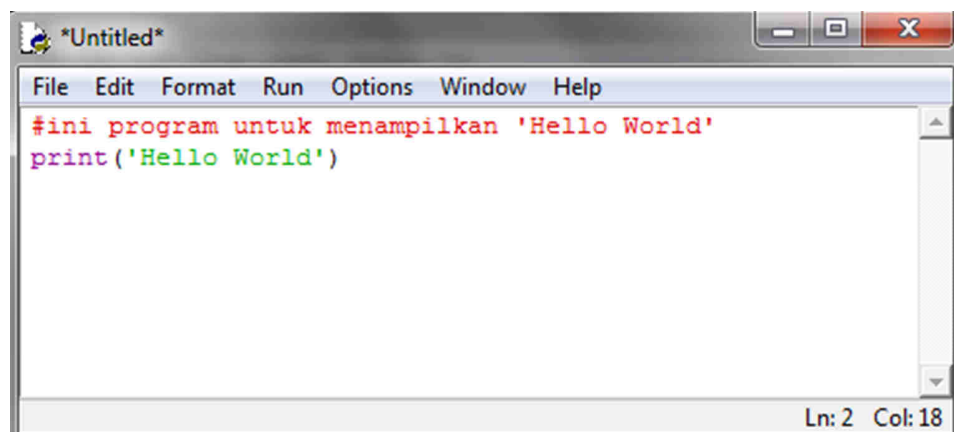
```

Dari contoh di atas tampak bahwa akan muncul error di dalam Python bahwa variabel `AlasSegitiga` belum didefinisikan sebelumnya (tidak dikenal).

## Membuat Program dengan Python

Setelah kita mengenal beberapa aturan dalam penulisan ekspresi, selanjutnya kita belajar bagaimana menuliskan kode program, menyimpannya ke dalam file, kemudian menjalankan programnya.

Untuk menuliskan kode program Python, kita bisa menggunakan Python IDLE dengan cara mengklik menu `FILE – NEW FILE`, kemudian ketikkan kode program yang diinginkan. Berikut ini contoh kode program untuk menampilkan teks 'Hello world' ke layar.



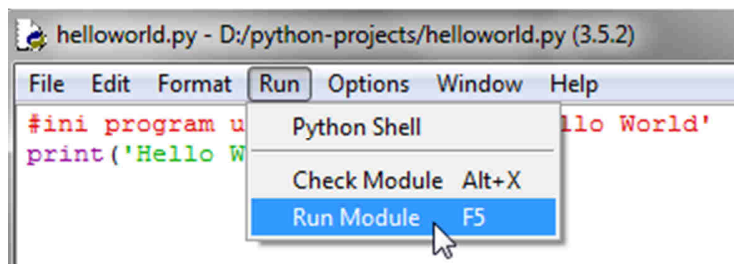
Keterangan:

- Perintah `print()` digunakan untuk mencetak teks yang terletak di dalam kurung
- Baris yang diawali dengan tanda `#` tidak akan diproses karena merupakan komentar

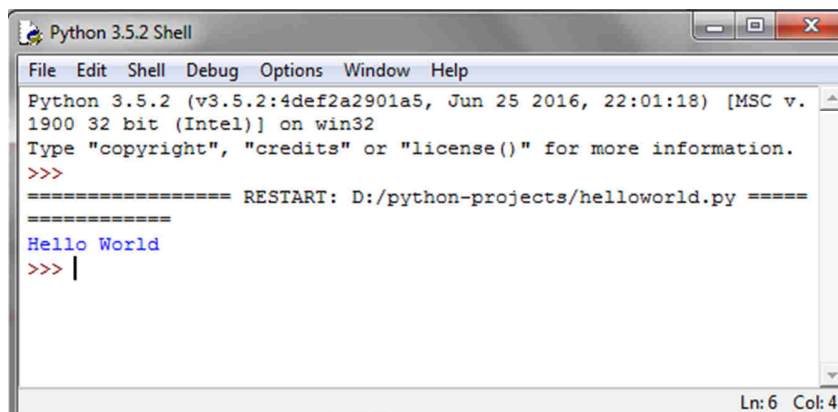
Setelah kode program dituliskan, program dapat disimpan dengan cara mengklik `FILE – SAVE`.

Kode program Python secara default disimpan dengan file berekstensi `.py`

Selanjutnya program dapat dijalankan dengan mengklik menu RUN – RUN MODULE



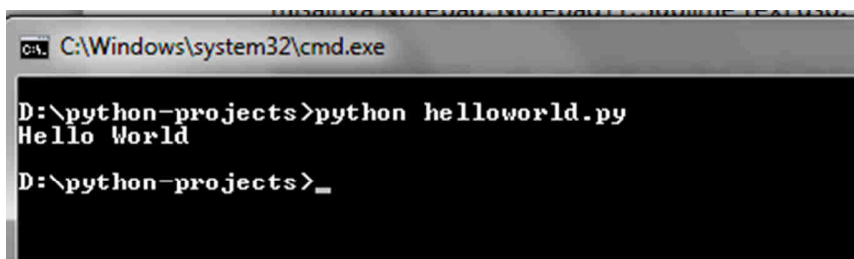
Adapun hasil dari running program, nanti akan muncul di Python Shell nya



Sebuah kode program Python yang dibuat dengan Python IDLE, dapat juga dijalankan melalui Command Prompt (console) dari sistem operasinya. Caranya adalah dengan masuk ke command prompt terlebih dahulu, kemudian masuk ke dalam direktori di mana file \*.py disimpan. Selanjutnya pada command prompt ketikkan perintah:

```
python [namafile].py
```

ketika diberikan perintah tersebut, akan muncul tampilan hasil dari program ketika dijalankan sebagaimana tampilan pada gambar berikut:

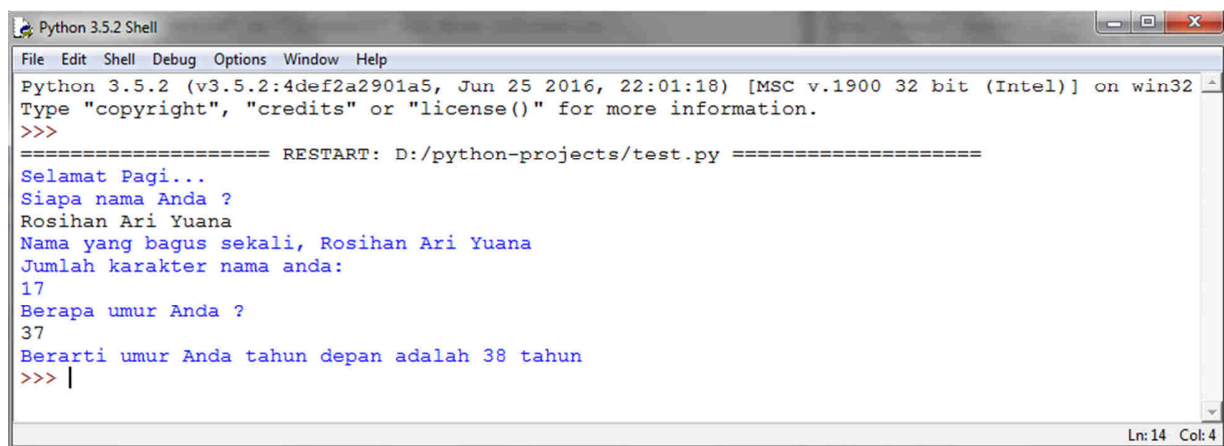


Setelah program 'Hello World' pertama sudah dibuat, selanjutnya kita mencoba membuat kode program yang kedua sbb (dengan terlebih dahulu mengklik FILE – NEW FILE):

```
# program menuliskan 'Selamat Pagi'
# kemudian menanyakan nama dan umur
# selanjutnya menampilkan jumlah karakter dari nama
# dan usia tahun depan

print('Selamat Pagi...')
print('Siapa nama Anda ? ');
yourName = input();
print('Nama yang bagus sekali, ' + yourName);
print('Jumlah karakter nama anda: ');
print(len(yourName));
print('Berapa umur Anda ? ');
yourAge = input()
print('Berarti umur Anda ' + str(int(yourAge) + 1) + ' tahun depan')
```

Jangan lupa menyimpan file kode programnya jika sudah selesai menuliskan, kemudian jalankan programnya. Adapun tampilan dari program setelah dijalankan adalah sbb:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/python-projects/test.py =====
Selamat Pagi...
Siapa nama Anda ?
Rosihan Ari Yuana
Nama yang bagus sekali, Rosihan Ari Yuana
Jumlah karakter nama anda:
17
Berapa umur Anda ?
37
Berarti umur Anda tahun depan adalah 38 tahun
>>> |
```

Melalui program tersebut, kita (sebagai user) bisa memasukkan nama kita sesukanya kemudian secara otomatis program akan menghitung jumlah karakter yang menyusun nama tersebut. Dalam hal ini, nama pada program yang kita buat dinamakan input. Demikian juga halnya dengan umur yang juga merupakan input. Jadi, input adalah data atau nilai yang merupakan masukan bagi program untuk diproses menghasilkan sebuah output. Adapun yang menjadi output dari program adalah jumlah karakter namanya dan sapaan 'Nama yang bagus sekali, [nama]' dan juga umur pengguna di tahun depan.

Sekarang, kita bahas beberapa perintah yang terdapat dalam kode program kedua tersebut.

- Perintah `input()` digunakan untuk membaca data/nilai yang dientrikan oleh user melalui perangkat keras input (mis. Keyboard)
  - Ekspresi `yourName = input()` adalah menyimpan data input yang diinputkan oleh user ke dalam variabel `yourName`
- NB: dalam hal ini, perintah data yang dibaca melalui `input()` selalu bertipe string.

- Perintah `print('Nama yang bagus sekali, ' + yourName)` digunakan untuk menampilkan string 'Nama yang bagus sekali' dan digabung dengan nilai string yang tersimpan dalam variabel `yourName`
- Perintah `print(len(yourName))`; digunakan untuk mencetak panjang karakter dari string yang tersimpan dalam variabel `yourName`. Untuk mendapatkan panjang karakter dari sebuah string, digunakan perintah `len()`.
- Ekspresi `yourAge = input()` digunakan untuk menyimpan input ke dalam variabel `yourAge`.
- Perintah `print('Berarti umur Anda tahun depan adalah ' + str(int(yourAge) + 1) + ' tahun')`  
Digunakan untuk mencetak string yang merupakan gabungan antara 3 string, yaitu 'Berarti umur Anda tahun depan adalah ', umur di tahun depan, dan ' tahun'. Dalam hal ini untuk mendapatkan umur di tahun depan bisa diperoleh dengan menjumlahkan umur yang diinputkan oleh user dengan 1.
- Perintah `int()` digunakan untuk mengkonversi suatu nilai ke tipe data bilangan bulat (integer). Pada program ini, perintah `int()` digunakan untuk mengkonversi nilai `yourAge` yang bertipe string ke integer supaya bisa dijumlahkan dengan 1.
- Perintah `str()` digunakan untuk mengkonversi suatu nilai ke tipe data string. Pada program ini, perintah `str()` digunakan untuk mengkonversi hasil penjumlahan `yourAge` dengan 1 ke tipe string supaya bisa digabung dengan string yang lainnya.

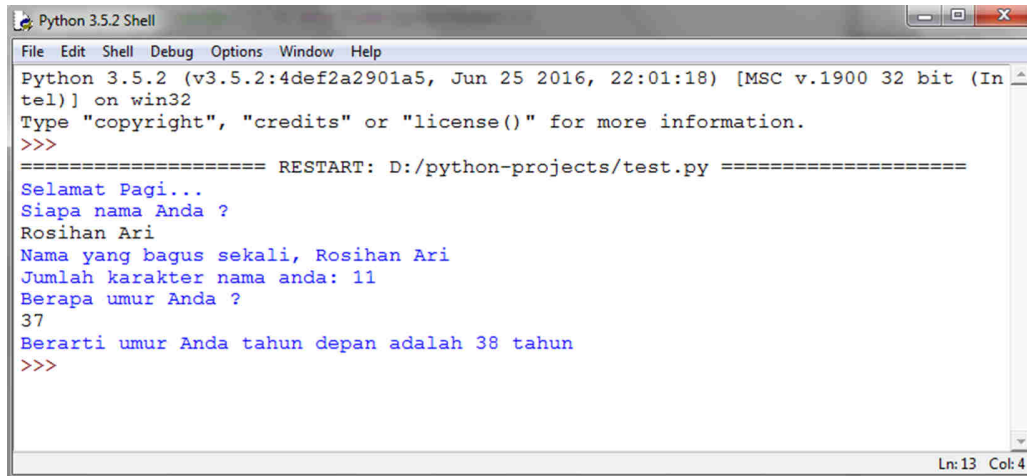
NB: untuk kasus lainnya, kita gunakan perintah `float()` untuk mengkonversi suatu nilai ke tipe data float (bilangan riil).

Setelah kita mengenal beberapa perintah untuk mengkonversi suatu nilai ke tipe data tertentu, maka kode program yang telah dibuat bisa dimodifikasi, khususnya pada bagian untuk menampilkan panjang karakter dari nama yang diinputkan menjadi sbb:

```
# program menuliskan 'Selamat Pagi'
# kemudian menanyakan nama dan umur
# selanjutnya menampilkan jumlah karakter dari nama
# dan usia tahun depan

print('Selamat Pagi...')
print('Siapa nama Anda ? ');
yourName = input();
print('Nama yang bagus sekali, ' + yourName);
print('Jumlah karakter nama anda: ' + str(len(yourName)));
print('Berapa umur Anda ? ');
yourAge = input();
print('Berarti umur Anda ' + str(int(yourAge) + 1) + ' tahun depan')
```

Dengan adanya perubahan tersebut, jumlah atau panjang karakter dari nama bisa dijadikan satu baris dengan string 'Jumlah karakter nama anda: '. Sehingga tampilan program setelah dijalankan menjadi



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/python-projects/test.py =====
Selamat Pagi...
Siapa nama Anda ?
Rosihan Ari
Nama yang bagus sekali, Rosihan Ari
Jumlah karakter nama anda: 11
Berapa umur Anda ?
37
Berarti umur Anda tahun depan adalah 38 tahun
>>>
```

## Statement Kontrol

Di dalam bahasa pemrograman terdapat mekanisme untuk mengatur aliran proses perintah yang dijalankan. Mekanisme ini disebut dengan statement kontrol. Secara umum terdapat 2 jenis statement kontrol, yaitu statement kontrol kondisional (bersyarat) dan perulangan (*looping*).

Sebelum membahas lebih dalam tentang statement kontrol, terlebih dahulu akan dibahas mengenai tipe data Boolean. Pembahasan tipe data Boolean ini sengaja terpisah dengan tipe data yang lainnya pada bab sebelumnya karena Boolean paling sering banyak digunakan dalam statement kontrol.

## Tipe Data Boolean

Tipe data Boolean adalah suatu tipe data yang hanya memiliki nilai True dan False.

Berikut ini contoh perintah untuk melakukan pemberian nilai bertipe Boolean ke dalam sebuah variabel (assignment):

```
myBoolean1 = True
myBoolean1
True
myBoolean2 = False
myBoolean2
False
```

Penulisan nilai True dan False harus benar karena bersifat case sensitif.

```
myBoolean = true
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    myBoolean = true
NameError: name 'true' is not defined
```

## Operator Tipe Data Boolean

Khusus untuk tipe data Boolean, terdapat beberapa operator sebagai berikut:

Operator	Makna
and	Dan
or	Atau
not	Tidak (negasi)

Hasil operasi menggunakan operator Boolean juga berupa nilai Boolean juga.

Berikut ini beberapa contoh penggunaan operator Boolean:

```
myBoolean = True or False
myBoolean
True
myBoolean = True and not False
myBoolean
True
```

```
myBoolean = not (True and True)
myBoolean
False
```

## Operator Relasional

Operator relasional digunakan untuk membandingkan dua buah nilai. Hasil dari penggunaan operator relasional berupa nilai Boolean. Operator ini berlaku untuk semua tipe data

Berikut ini beberapa operator relasional yang bisa digunakan:

Operator	Makna
==	Sama dengan
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
!=	Tidak sama dengan

Adapun contoh-contoh penggunaan operator relasional antara lain:

```
2 == 3
False
```

```
3 != 4
```



```
True
```

```
(2 > 3) and (3 == 5)
False
```

```
42 == '42'
False
```

```
'hallo' == 'hallo'
True
```

```
'Hallo' == 'HALLO'
False
```

```
bil = 50
bil < 40
False
```

## Statement Kondisional - IF

Setelah mengenal tipe data Boolean dan operator Boolean serta operator relasional, selanjutnya kita mulai membahas tentang statement kondisional.

Statement kondisional digunakan untuk menyatakan pernyataan bersyarat. Di dalam Python hanya terdapat 1 statemen yang menyatakan kondisional, yaitu IF dengan sintaks:

```
if syarat :
    ekspresi...
    ekspresi...
```

Pada sintaks tersebut, *syarat* harus bernilai boolean (True atau False). Jika *syarat* bernilai True, maka *ekspresi* akan dijalankan. Sedangkan jika bernilai False maka tidak melakukan apa-apa.

*Catatan: Sebagai penanda blok ekspresi yang akan dilakukan ketika syarat bernilai True adalah adanya indentasi (tab).*

Adapun contohnya adalah:

```
bil = 30
if bil > 10 :
    print('lebih besar dari 10')
```

Ketika kode program di atas dijalankan, maka program akan mencetak 'lebih besar dari 10'

Pada contoh yang diberikan, nilai variabel `bil` adalah 30. Sehingga dalam hal ini, syarat `bil > 10` bernilai `True` dan akibatnya akan mencetak 'lebih besar dari 10'.

Perhatikan contoh selanjutnya berikut ini:

```
bil = 20
if bil > 30 :
    print('lebih besar dari 30')
```

Hasil dari contoh di atas tidak menampilkan apa-apa ketika dijalankan, karena nilai syarat nya adalah `False` sehingga tidak ada ekspresi yang akan dijalankan.

Statement IF dapat pula berbentuk sebagaimana sintaks berikut ini:

```
if syarat :
    ekspresi1
else :
    ekspresi2
```

Bentuk IF dengan sintaks tersebut, `ekspresi2` akan dijalankan ketika syarat bernilai `False`. Sedangkan `ekspresi1` akan dijalankan ketika syarat bernilai `True`.

Dalam hal ini, penanda blok ekspresi yang akan dieksekusi ketika syarat bernilai `True` atau ketika syarat bernilai `False` adalah dengan adanya indentasi (tab).

Adapun contoh penggunaannya adalah:

```
bil = 20
if bil < 10:
    print('lebih kecil dari 10')
else:
    print('tidak lebih kecil dari 10')
```

Ketika kode program tersebut dijalankan, maka akan dihasilkan tampilan 'tidak lebih kecil dari 10'. Hal ini disebabkan karena syarat bernilai `False`, sehingga yang dijalankan adalah ekspresi pada `else`.

Selain dua bentuk statement IF yang telah diberikan, terdapat juga bentuk statement IF dengan sintaks:

```
if syarat1 :
    ekspresi1
elif syarat2 :
```

```

    ekspresi2
elif syarat3 :
    ekspresi3
.
.
else :
    ekspresiN

```

Berdasarkan sintaks di atas, apabila syarat1 bernilai True maka ekspresi1 yang akan dijalankan, sedangkan ekspresi2, ekspresi3 dst tidak akan dijalankan. Namun, jika syarat1 bernilai False maka akan dilakukan pengecekan untuk syarat2. Apabila syarat2 bernilai True maka ekspresi2 yang akan dijalankan. Sedangkan apabila syarat2 bernilai False maka akan dilakukan pengecekan pada syarat3 dst. Adapun apabila tidak ada syarat yang bernilai True, maka ekspresi yang akan dijalankan ekspresiN.

Berikut ini contoh penggunaannya:

```

bil = 10
if bil < 10:
    print('lebih kecil dari 10')
elif bil > 10:
    print('lebih besar dari 10')
else:
    print('sama dengan 10')

```

Apabila program di atas dijalankan, maka tampilan yang muncul adalah 'sama dengan 10'. Hal ini disebabkan karena kedua syarat sebelumnya semuanya bernilai False.

Sedangkan untuk contoh penggunaan berikut ini:

```

bil = 10
if bil < 10:
    print('lebih kecil dari 10')
elif bil == 10:
    print('sama dengan 10')
else:
    print('lebih kecil dari 10')

```

akan diperoleh hasil tampilan 'sama dengan 10' yang dihasilkan pada saat syarat ke-2 (bil == 10) bernilai True.

### Studi Kasus

Buatlah program Python untuk mengkonversi nilai angka 0-100 ke dalam bentuk nilai huruf, dengan ketentuan sebagai berikut

Nilai Angka	Nilai Huruf
80 – 100	A
70 – 79	B
60 – 69	C
40 – 59	D
0 – 39	E

### Jawab:

Berdasarkan kasus yang diberikan, maka untuk kasus ini terdapat satu buah input yaitu nilai angka. Adapun outputnya adalah nilai huruf. Sehingga solusi programnya adalah sebagai berikut:

```
# program konversi nilai angka 0-100
# ke nilai huruf

print('Masukkan nilai angka : ')
nAngka = int(input())

if nAngka >= 80 and nAngka <=100:
    nHuruf = 'A'
elif nAngka >= 70 and nAngka <= 79:
    nHuruf = 'B'
elif nAngka >= 60 and nAngka <= 69:
    nHuruf = 'C'
elif nAngka >= 40 and nAngka <= 59:
    nHuruf = 'D'
elif nAngka >= 0 and nAngka <= 39:
    nHuruf = 'E'
else:
    print('Nilai angka tidak valid')

print('Nilai hurufnya adalah : ', nHuruf)
```

Program tersebut apabila dijalankan dan kemudian diberikan input angka 0 s/d 100 maka bisa memberikan hasil yang benar. Misalkan apabila diberikan input angka 50 maka akan memberikan output nilai hurufnya D

```
Masukkan nilai angka :
50
Nilai hurufnya adalah : D
```

Namun, apabila dimasukkan nilai angka di luar 0 s/d 100, misalkan 200 maka akan muncul error sebagai berikut

```
Masukkan nilai angka :
200
Nilai angka tidak valid
Traceback (most recent call last):
  File "D:\python projects\konversi.py", line 20, in <module>
    print('Nilai hurufnya adalah : ', nHuruf)
NameError: name 'nHuruf' is not defined
```

Error tersebut muncul dikarenakan kegagalan pada proses menjalankan perintah `print()`. Di dalam perintah `print()` terdapat `nHuruf` yang akan dicetak, sedangkan untuk kasus nilai angka 200 nilai `nHuruf` tidak ada. Oleh karena itu sebaiknya program tersebut dimodifikasi, misalnya menjadi berikut ini

```
# program konversi nilai angka 0-100
# ke nilai huruf

print('Masukkan nilai angka : ')
nAngka = int(input())

if nAngka >= 80 and nAngka <=100:
    nHuruf = 'A'
elif nAngka >= 70 and nAngka <= 79:
    nHuruf = 'B'
elif nAngka >= 60 and nAngka <= 69:
    nHuruf = 'C'
elif nAngka >= 40 and nAngka <= 59:
    nHuruf = 'D'
elif nAngka >= 0 and nAngka <= 39:
    nHuruf = 'E'
else:
    print('Nilai angka tidak valid')

#modifikasi pada bagian cetak nilai huruf
if nAngka >= 0 and nAngka <= 100:
    print('Nilai hurufnya adalah : ', nHuruf)
```

Modifikasi yang dilakukan pada program tersebut adalah dengan memberi kondisional (syarat) pada proses `print()` yaitu hanya akan dilakukan apabila nilai angkanya 0 s/d 100 saja.

Alternatif lain dalam memodifikasi program, dapat pula menjadi berikut ini:

```
# program konversi nilai angka 0-100
# ke nilai huruf
```

```

print('Masukkan nilai angka : ')
nAngka = int(input())

if nAngka >= 0 and nAngka <= 100:
    if nAngka >= 80 and nAngka <= 100:
        nHuruf = 'A'
    elif nAngka >= 70 and nAngka <= 79:
        nHuruf = 'B'
    elif nAngka >= 60 and nAngka <= 69:
        nHuruf = 'C'
    elif nAngka >= 40 and nAngka <= 59:
        nHuruf = 'D'
    elif nAngka >= 0 and nAngka <= 39:
        nHuruf = 'E'

    print('Nilai hurufnya adalah : ', nHuruf)
else:
    print('Nilai angka tidak valid')

```

Modifikasi alternatif lain pada program di atas dilakukan dengan memasukkan perintah IF untuk menentukan nilai hurufnya ke dalam syarat if `nAngka >= 0 and nAngka <= 100`. Artinya bahwa proses penentuan nilai angka hanya akan dilakukan ketika nilai angkanya 0 s/d 100 saja. Sedangkan apabila tidak, maka nilai angkanya tidak valid.



## Statement Perulangan – WHILE

Statement perulangan digunakan untuk mengulang ekspresi atau perintah hingga mencapai batas tertentu. Dalam hal ini, batas perulangan bisa berupa banyaknya perulangan, artinya bahwa perulangan akan berhenti apabila banyaknya perulangan yang sudah dilakukan telah mencapai banyaknya perulangan yang diinginkan. Selain itu batas perulangan juga bisa berupa syarat kapan perulangan harus berhenti. Analogi kedua jenis batas perulangan tersebut adalah pada dua kalimat berikut:

1. Ali makan bakso sampai dengan 10 kali
2. Ali terus makan bakso selama dia masih belum kenyang

Kalimat nomor 1, perulangan yang dilakukan oleh Ali adalah makan bakso. Proses makan bakso yang dilakukan oleh Ali baru akan berhenti apabila dia telah makan 10 kali. Sehingga dalam contoh ini perulangan akan berhenti ketika telah mencapai jumlah perulangan tertentu yang diinginkan. Sedangkan pada kalimat nomor 2, proses makan yang dilakukan oleh Ali tidak diketahui banyak perulangannya. Dalam hal ini, yang diketahui kapan dia berhenti makan bakso yaitu ketika dia merasa kenyang. Artinya selama Ali masih belum merasa kenyang maka dia akan terus makan bakso.

Statement WHILE dapat digunakan untuk kedua jenis keadaan perulangan di atas. Dengan kata lain, WHILE dapat digunakan untuk perulangan yang diketahui banyaknya perulangan atau tidak (hanya diketahui syarat selama apa perulangan tersebut dilakukan).

Struktur sintaks dari WHILE adalah sbb:

```
while syarat:
    ekspresi...
    ekspresi...
```

Berdasarkan sintaks tersebut, selama syarat bernilai `True` maka ekspresi dalam blok while akan terus dilakukan. Dengan kata lain, perulangan while baru akan berhenti ketika syarat bernilai `False`.

Berikut ini adalah contoh penggunaannya:

```
n = 1
while n <= 100:
    print('Ali makan bakso')
    n = n + 1
```

Apabila kode program di atas dijalankan maka akan menghasilkan tampilan

```
Ali makan bakso
Ali makan bakso
Ali makan bakso
```

Kode program tersebut mencetak 'Ali makan bakso' sebanyak 3 kali. Adapun penjelasan dari contoh kode tersebut adalah sebagai berikut:

Nilai awal untuk variabel `n` adalah 1. Variabel `n` di sini dimaksudkan sebagai penghitung banyaknya perulangan yang akan dilakukan. Adapun syarat perulangan adalah  $n \leq 3$ , yang artinya bahwa selama nilai  $n \leq 3$  maka perulangan akan tetap terus dilakukan. Ekspresi `n = n + 1` digunakan untuk menambah nilai `n` sebelumnya. Sehingga proses yang terjadi ketika kode program dijalankan adalah

```
n = 1
apakah n ≤ 3? → True
    cetak 'Ali makan bakso'
    n = n + 1 → n = 1 + 1 = 2
apakah n ≤ 3? → True
    cetak 'Ali makan bakso'
    n = n + 1 → n = 2 + 1 = 3
apakah n ≤ 3? → True
    cetak 'Ali makan bakso'
    n = n + 1 → n = 3 + 1 = 4
apakah n ≤ 3? → False
```

Contoh WHILE yang diberikan di atas merupakan perulangan yang diketahui banyaknya perulangan.

Selanjutnya diberikan contoh kode program berikut ini:

```
bil = 0
while bil != -1:
    print('Masukkan sebuah bilangan :')
    bil = int(input())
```

Kode program tersebut ketika dijalankan akan terus meminta input bilangan selama bilangan yang diinputkan bukan -1. Dalam hal ini banyaknya perulangan tidak diketahui, namun hanya kapan syarat perulangan itu berhenti yaitu selama inputnya bukan -1.

Penggunaan ekspresi `bil = 0` sebelum `WHILE` adalah untuk nilai awal supaya perulangan `WHILE` nya bisa berjalan. Nilai awal yang dipilih bisa sembarang nilai asalkan bukan -1.

Adapun output dari contoh kode di atas adalah:

```
Masukkan sebuah bilangan :
9
Masukkan sebuah bilangan :
1
Masukkan sebuah bilangan :
2
Masukkan sebuah bilangan :
-1
```

### Statement *'break'*

Sebuah perulangan dapat dihentikan meskipun syarat perulangan masih memenuhi (bernilai `True`) dengan menggunakan statement `break`. Berikut ini adalah contoh penggunaannya:

```
while True:
    print('Masukkan bilangan : ')
    bil = int(input())
    if bil == -1:
        break
print('Terimakasih')
```

Pada contoh di atas, perulangan `WHILE` selalu akan terus dilakukan karena syaratnya adalah `True`. Namun jika input bilangannya adalah -1 maka perulangannya akan berhenti. Selanjutnya setelah `WHILE` berhenti akan mencetak 'Terimakasih'.

### Statement *'continue'*

Di dalam perulangan bisa terdapat statement `continue`. Statement ini digunakan untuk mengulangi kembali ke bagian awal looping tanpa menyelesaikan satu perulangan utuh. Perhatikan contoh berikut ini:

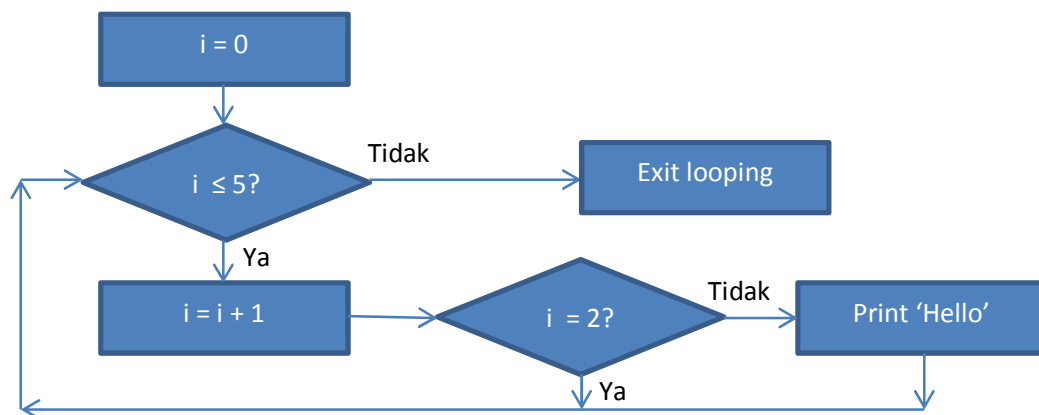


```
i = 0;
while i<=5:
    i = i+1
    if i==2:
        continue
    print('i=' + str(i) + ', Hello')
```

Apabila kode tersebut dijalankan maka akan muncul output:

```
i=1, Hello
i=3, Hello
i=4, Hello
i=5, Hello
i=6, Hello
```

Nilai *i* mula-mula (sebelum masuk ke while adalah 0). Selanjutnya di dalam while, perulangan akan dilakukan selama nilai  $i \leq 5$ . Di dalam looping nilai *i* bertambah 1 dari sebelumnya, kemudian mencetak 'hello'. Namun ketika  $i=2$ , diberikan statement continue sehingga pengaruhnya adalah ketika  $i=2$  tidak dilakukan menuliskan 'hallo' karena setelah continue aliran proses program kembali ke awal looping. Secara diagram proses ini digambarkan sebagai berikut:



### Studi Kasus

Buatlah sebuah program untuk menjumlahkan 2 buah bilangan bulat yang berasal dari input. Setiap kali menampilkan hasil penjumlahannya, munculkan pertanyaan (yes/no) apakah user ingin mengulangnya lagi. Jika user memilih yes maka program meminta user memasukkan kembali kedua bilangan yang akan dijumlahkan kemudian program menampilkan hasilnya. Namun jika user memilih no, maka program berhenti.

**Jawab:**

Pada studi kasus yang diberikan, terdapat perulangan yang banyaknya perulangan tidak ditentukan. Perulangan akan terus dilakukan selama user memilih yes untuk mengulangi penjumlahan kedua bilangan. Sehingga kode program yang menjadi solusi dari studi kasus ini adalah

```
jawab = 'y'
while jawab == 'y':
    print('Masukkan bil pertama : ')
    bil1 = int(input())
    print('Masukkan bil kedua : ')
    bil2 = int(input())
    hasil = bil1 + bil2
    print('Hasil penjumlahannya : ', hasil)
    print('Ulangi lagi (y/n)? : ')
    jawab = input()
```

Dalam kode program tersebut, yang merupakan syarat bagi perulangan WHILE adalah `jawab == 'y'` atau dengan kata lain selama jawaban dari user ketika ditanyakan akan mengulangi kembali atau tidak adalah yes. Oleh karena itu, perlu diberikan nilai awal variabel `jawab` sebelum WHILE yaitu `'y'` (`jawab = 'y'`) supaya perulangan WHILE dapat berjalan ketika pertama kali. Selanjutnya nilai `jawab` tergantung input yang dilakukan user.

Sehingga output dari program di atas adalah:

```
Masukkan bil pertama :
4
Masukkan bil kedua :
5
Hasil penjumlahannya : 9
Ulangi lagi (y/n)? :
y
Masukkan bil pertama :
5
Masukkan bil kedua :
8
Hasil penjumlahannya : 13
Ulangi lagi (y/n)? :
y
Masukkan bil pertama :
2
Masukkan bil kedua :
1
Hasil penjumlahannya : 3
Ulangi lagi (y/n)? :
n
```



### Studi Kasus

Buatlah kode program untuk mengenerate syair lagu ‘anak ayam’ sebagai berikut

*Anak ayam turun 3, mati satu tinggallah 2*

*Anak ayam turun 2, mati satu tinggallah 1*

*Anak ayam turun 1, mati satu tinggal induknya*

Adapun jumlah anak ayam mula-mula adalah  $n$  buah yang merupakan input program. Contoh syair anak ayam di atas adalah untuk  $n = 3$ .

#### Jawab:

Studi kasus yang diberikan di atas terdapat perulangan yaitu mencetak ‘*Anak ayam turun ..., mati satu tinggallah ...*’. Perulangan tersebut diulang sebanyak  $n$  kali, namun ada pengecualian ketika jumlah anaknya satu. Selain itu, jumlah anak ayam ( $n$ ) pada setiap kali perulangan nilainya berkurang satu. Sehingga kode program yang bisa menjadi solusi dari studi kasus ini adalah:

```
print('Masukkan jumlah anak ayam : ')
n = int(input())
while n >= 1:
    if n > 1:
        print('Anak ayam turun', n, ', mati satu tinggallah ', n-1)
    else:
        print('Anak ayam turun', n, ', mati satu tinggal induknya')
    n = n - 1
```

Dalam kode program di atas, perulangan WHILE diberikan syarat  $n \geq 1$  yang artinya bahwa perulangan akan terus dilakukan selama jumlah anak ayam ( $n$ ) masih  $\geq 1$ . Selanjutnya di dalam WHILE terdapat sebuah kondisi di mana akan mencetak ‘*Anak ayam turun  $n$ , mati satu tinggallah  $n-1$* ’ atau ‘*Anak ayam turun  $n$ , mati satu tinggal induknya*’. Sesuai dengan kasusnya, maka perintah untuk mencetak ‘*Anak ayam turun  $n$ , mati satu tinggallah  $n-1$* ’ dilakukan ketika anak ayamnya lebih dari 1. Namun ketika  $n = 1$  maka mencetak ‘*Anak ayam turun  $n$ , mati satu tinggal induknya*’. Adapun ekspresi  $n = n - 1$  digunakan untuk mengurangi nilai  $n$  sebelumnya. Sehingga output dari kode program di atas adalah

```
Masukkan jumlah anak ayam :
3
Anak ayam turun 3 , mati satu tinggallah 2
Anak ayam turun 2 , mati satu tinggallah 1
Anak ayam turun 1 , mati satu tinggal induknya
```



## Infinite Loop

Di dalam sebuah perulangan, bisa jadi dikarenakan kesalahan logika sehingga mengakibatkan perulangan terus menerus dilakukan tanpa berhenti (*infinite loop*). Sebagai contoh adalah kode program berikut ini

```
n = 1
while n < 5:
    print('Hello world')
```

Apabila kode di atas dijalankan maka perulangan dalam mencetak 'Hello world' tidak akan berhenti dikarenakan nilai *n* (yaitu 1) selalu memenuhi *n* < 5. Jika terjadi *infinite loop* di dalam Python, maka cara untuk menghentikan perulangan adalah dengan menekan tombol CTRL + C, atau bisa dengan cara lain yaitu merestart shell dengan mengklik menu SHELL > RESTART SHELL (atau menekan CTRL + F6). Selanjutnya kode program harus dimodifikasi supaya tidak terjadi *infinite loop* kembali.

## Statement Perulangan – FOR

Statement perulangan berikutnya yang akan dibahas adalah FOR. Statement FOR digunakan untuk menyatakan perulangan yang sudah pasti banyaknya perulangan.

Sintaks dari perulangan FOR adalah sebagai berikut

```
for i in range(n):
    ...
    ...
```

Berdasarkan sintaks di atas, perulangan FOR dilakukan sebanyak *n* kali. Dalam hal ini, *i* merupakan sebuah variabel yang berfungsi sebagai pencacah (*counter*). Untuk setiap perulangan, nilai *i* ini akan berubah nilainya mulai dari 0, 1, 2, ..., *n*-1 (total *n* perulangan). Setelah *i* nilainya sama dengan *n*-1 maka perulangan berhenti.

Perhatikan contoh berikut ini:

```
for i in range(4):
    print('i=' + str(i) + ', Hello world')
```

Contoh di atas akan menghasilkan output:

```
i=0, Hello world
i=1, Hello world
i=2, Hello world
i=3, Hello world
```

Berdasarkan contoh tersebut tampak bahwa nilai *i* berubah pada setiap perulangannya mulai dari 0 sampai dengan 3 atau dalam hal ini sebanyak 4 kali perulangan. Selain mencetak nilai *i*, di dalam perulangan juga mencetak 'Hello world'.

Penggunaan `range(n)` pada perulangan `for` dapat dalam bentuk yang lain. Kita juga bisa menggunakan `range(a, n)`. Misalnya pada program berikut ini

```
for i in range(1, 5):
    print(i)
```

Output dari program tersebut adalah sebagai berikut:

```
1
2
3
4
```

Sehingga dalam hal ini, penggunaan `range(a, b)` akan memberi nilai variabel counternya mulai dari *a*, *a*+1, *a*+2, ... dan berhenti hingga sebelum *b*.

Selain itu, `range()` dapat pula berbentuk `range(a, b, step)`. Misalnya adalah:

```
for i in range(1, 15, 2):
    print(i)
```

Program tersebut jika dijalankan akan dihasilkan output:

```
1
3
5
7
9
11
13
```

Berdasarkan output tersebut, dapat disimpulkan bahwa nilai `step` di dalam `range(a, b, step)` memiliki makna besarnya kenaikan nilai variabel counternya. Mula-mula nilai variabel counternya adalah *a*, kemudian setiap kali perulangan nilainya bertambah sejumlah `step`, dan perulangan baru akan berhenti hingga nilainya sebelum mencapai *b*.

### Studi Kasus

Buatlah program untuk menampilkan bilangan bulat 1 s/d 100 yang merupakan kelipatan 3.

**Jawab:**

Untuk menyelesaikan kasus di atas, kita bisa menggunakan perulangan FOR dengan membuat nilai  $i = 0, 1, 2, \dots, 99$ . Selanjutnya karena pada kasus yang diberikan bilangan bulatnya mulai dari 1, maka untuk setiap  $i$  tersebut perlu ditambah 1 supaya diperoleh bilangan 1, 2, 3, ..., 100. Kemudian setiap dari bilangan yang diperoleh ini dicek apakah merupakan kelipatan 3 atau tidak. Jika ya, maka ditampilkan. Sedangkan jika tidak, maka diabaikan. Sehingga dari ide ini diperoleh program sebagai berikut:

```
for i in range(100):
    bil = i+1
    if bil % 3 == 0:
        print(bil)
```

Alternatif lain, kita bisa juga menuliskan kode programnya sebagai berikut

```
for i in range(100):
    if (i+1) % 3 == 0:
        print(i+1)
```



### Studi Kasus

Buatlah program untuk menghitung banyaknya bilangan bulat 1 s/d 100 yang merupakan kelipatan 3.

#### Jawab:

Kasus ini merupakan pengembangan dari kasus sebelumnya. Ide untuk menghitung banyaknya bilangan bulat yang merupakan kelipatan 3 ini adalah dengan membuat proses perhitungan yang nilainya bertambah 1 (*increment* dengan faktor 1) setiap kali ditemukan bilangan yang merupakan kelipatan 3. Sebelum mulai perhitungan, terlebih dahulu diberikan nilai awal 0.

```
jumlah = 0
for i in range(100):
    if (i+1) % 3 == 0:
        jumlah = jumlah + 1
print('Banyaknya bilangan : ' + str(jumlah))
```



### Studi Kasus

Buatlah program untuk menghitung hasil dari  $1 + 2 + 3 + \dots + 100$ .

#### Jawab:

Di dalam program, kasus di atas dapat diselesaikan dengan menggunakan looping. Ide penyelesaiannya adalah sebagai berikut:

- Mula-mula sebelum mulai menjumlahkan, kita beri nilai awal variabel `sum = 0`. Variabel `sum` ini digunakan untuk menyimpan nilai hasil penjumlahannya.
- Selanjutnya dimulai proses penjumlahannya satu persatu untuk setiap suku yang dijumlahkan dengan cara nilai `sum` sebelumnya dijumlahkan dengan suku pertama dan hasilnya disimpan kembali ke dalam `sum`

$$\text{sum} = \text{sum} + 1 = 0 + 1 = 1$$

- Selanjutnya dilakukan lagi penjumlahan nilai `sum` dengan suku kedua, dan hasilnya disimpan ke dalam `sum` lagi.

$$\text{sum} = \text{sum} + 2 = 1 + 2 = 3$$

- Proses penjumlahan ini dilakukan terus hingga semua sukunya terjumlahkan.

Berdasarkan ide penyelesaian di atas, maka proses di atas bisa dinyatakan ke dalam bentuk looping FOR karena banyaknya perulangan di atas sudah jelas. Sehingga kode program penyelesaiannya adalah

```
sum = 0
for i in range(100):
    suku = i+1
    sum = sum + suku
print('Hasil penjumlahannya : ' + str(sum))
```

Dalam hal ini nilai `i` berubah dimulai dari 0, 1, 2, ..., 99. Sedangkan karena suku yang dijumlahkan 1, 2, 3, ..., 100 maka nilai `i` perlu ditambahkan 1 terlebih dahulu (`suku = i + 1`) baru kemudian dijumlahkan.

Selain menggunakan kode program di atas, kita bisa juga menyelesaikan kasusnya dengan looping sebanyak 101 kali sebagai berikut:

```
sum = 0
for i in range(101):
    sum = sum + i
print('Hasil penjumlahannya : ' + str(sum))
```

Jika menggunakan kode tersebut, maka looping terjadi untuk `i = 0` sampai dengan 100, dengan kata lain akan menghitung penjumlahan  $0 + 1 + 2 + \dots + 100$ . Sama saja bukan??

Adapun output dari kedua program di atas adalah

```
Hasil penjumlahannya : 5050
```



### Studi Kasus

Buatlah program untuk menampilkan semua bilangan yang habis dibagi 3 dan 5 antara 1 sampai 100, kemudian hitung banyaknya bilangan, serta menghitung jumlahan semua bilangan tersebut.

#### Jawab:

Ide penyelesaian dari kasus yang diberikan adalah menggunakan perulangan FOR, dimulai dari  $i = 0$  sampai dengan 99 (100 bilangan). Sehingga bilangan 1 s/d 100 dapat diperoleh dengan  $bil = i + 1$ . Di antara nilai  $bil$  tersebut, nanti dilakukan pengecekan apakah  $bil$  tersebut habis dibagi 3 dan 5 atau tidak. Jika ya, maka masuk hitungan dan dijumlahkan, serta ditampilkan. Sedangkan jika tidak, maka diabaikan. Untuk menghitung banyaknya bilangan dengan menggunakan cara menambahkan 1 dari hasil hitungan sebelumnya setiap kali ditemukan bilangan yang habis dibagi 3 dan 5.

Sehingga kode program untuk solusi kasus tersebut adalah sebagai berikut:

```
hitung=0
sum=0
for i in range(100):
    bil = i+1
    if bil%3==0 and bil%5==0:
        hitung = hitung + 1
        sum = sum + bil
        print(bil)

print('Banyak bilangan    : ' + str(hitung))
print('Jumlahan bilangan : ' + str(sum))
```

Adapun output dari kode program di atas adalah

```
15
30
45
60
75
90
Banyak bilangan    : 6
Jumlahan bilangan : 315
```

■

### Ekuivalensi Perulangan WHILE dan FOR

Seperti yang pernah dibahas sebelumnya bahwa statement WHILE dapat digunakan untuk menyatakan perulangan yang diketahui banyaknya perulangan. Demikian juga halnya dengan statement FOR. Oleh karena itu apabila dijumpai kasus di mana terdapat perulangan yang demikian, maka kita bisa menggunakan salah satu dari keduanya.



Perhatikan kedua kode program berikut ini untuk mencetak 'Hello World' sebanyak 5 kali, di mana program pertama menggunakan FOR, sedangkan yang ke dua menggunakan WHILE.

```
for i in range(5):
    print('i=' + str(i) + ', Hello world')
```

```
i = 0
while i<5:
    print('i=' + str(i) + ', Hello world')
    i = i + 1
```

Jika kedua program di atas dijalankan, maka akan menghasilkan output yang sama yaitu

```
i=0, Hello world
i=1, Hello world
i=2, Hello world
i=3, Hello world
i=4, Hello world
```

Meskipun keduanya menghasilkan output yang sama, namun jika dilihat dari efisiensi penulisan kode program tampak bahwa penulisan FOR lebih simpel dibandingkan WHILE. Apabila menggunakan WHILE untuk menyatakan perulangan yang diketahui banyaknya perulangan butuh penulisan nilai awal, dan ekspresi increment ( $i = i + 1$ ). Sedangkan di dalam FOR kedua hal ini tidak perlu dituliskan secara eksplisit karena keduanya sudah terkandung secara implisit di dalam sintaks FOR itu sendiri. Oleh karena itu, statement FOR lebih disarankan digunakan untuk menyatakan perulangan yang diketahui banyaknya perulangan.

### Soal-soal Latihan

1. Buatlah program untuk menghitung banyaknya bilangan ganjil antara 2 s/d 230, beserta jumlahan seluruh bilangannya
2. Buatlah program untuk menampilkan formasi bintang yang ditentukan berdasarkan nilai  $n$ :

Contoh untuk  $n = 3$

```
*
* *
* * *
```

Contoh untuk  $n = 4$

```
*
* *
* * *
* * * *
```

Keterangan:

- Nilai  $n$  ditentukan melalui input
  - Program dibuat dengan menggunakan FOR dan WHILE
3. Buatlah program untuk menampilkan formasi bintang yang ditentukan berdasarkan nilai  $n$ :

Contoh untuk  $n = 3$

```
* * *
* *
*
```

Contoh untuk  $n = 4$

```
* * * *
* * *
* *
*
```

Keterangan:

- Nilai  $n$  ditentukan melalui input
  - Program dibuat dengan menggunakan FOR dan WHILE
4. Buatlah program untuk menampilkan formasi bintang yang ditentukan berdasarkan nilai  $n$ :

Contoh untuk  $n = 3$

```
*
* *
* * *
```

Contoh untuk  $n = 4$

```
*
* *
* * *
* * * *
```

Keterangan:

- Nilai  $n$  ditentukan melalui input
- Program dibuat dengan menggunakan FOR dan WHILE

5. Buatlah program untuk menghitung hasil dari

$$\frac{1}{2} - \frac{2}{3} + \frac{3}{4} - \frac{4}{5} + \dots + \frac{n}{n+1}$$

Nilai  $n$  ditentukan dari input.

6. Buatlah program untuk menghitung besar pulsa telepon apabila diberikan tabel biaya pulsa untuk setiap daerah sebagai berikut:

Kode Wilayah	Nama Wilayah	Tarif Pulsa	Lama per pulsa
0271	Surakarta	Rp 750	1 menit

021	DKI Jakarta	Rp 1000	30 detik
0274	DIY	Rp 800	25 detik
024	Semarang	Rp 850	20 detik
031	Surabaya	Rp 1100	17 detik

Keterangan:

Setiap wilayah, perhitungan satuan waktu per pulsa berbeda-beda seperti pada tabel

Input dari program adalah sebagai berikut:

- Kode wilayah
- Waktu mulai percakapan telepon yang terdiri dari: jam, menit, detik
- Waktu selesai percakapan telepon yang terdiri dari: jam, menit, detik

Adapun outputnya adalah:

- Nama Wilayah : XXXXX (Kode: XXX)
- Waktu Mulai Telepon (jam: menit:detik) : XX:XX:XX
- Waktu Selesai Telepon (jam: menit:detik) : XX:XX:XX
- Total Waktu : XXX detik
- Total Biaya Pulsa : Rp XXXXX

7. Buatlah program untuk membaca input beberapa bilangan riil. Data bilangan riil ini dilakukan berulang-ulang. Setiap sehabis menginput bilangan, user diberikan pertanyaan apakah mau menginput lagi. Setelah user memilih untuk tidak lagi mengulang input, selanjutnya program menampilkan rata-rata semua bilangan yang diinputkan tadi.

Contoh tampilan program ketika dijalankan:

```

Masukkan bilangan : XXX
Ulangi lagi (y/n) ? y
Masukkan bilangan : XXX
Ulangi lagi (y/n) ? y
Masukkan bilangan : XXX
Ulangi lagi (y/n) ? y
Masukkan bilangan : XXX
Ulangi lagi (y/n) ? n

```

Rata-rata semua bilangannya adalah : XXX

8. Buatlah program untuk simulasi interface sistem security suatu aplikasi berupa login. Dalam hal ini, user diminta memasukkan username dan password yang sudah ditentukan. User hanya diberikan kesempatan mengulang loginnya jika salah maksimal 3 kali. Jika loginnya sukses, maka akan muncul kalimat 'Selamat datang XXX, Anda telah berhasil login'. Jika loginnya telah gagal 3 kali, maka program berhenti.

Sebagai contoh, misalkan username yang telah diset adalah 'rosihanari' dengan password '123', maka berikut ini tampilan programnya jika 3 kali salah login:

```
Masukkan Username : amir
Masukkan Password : hehehe
Masukkan Username : budi
Masukkan Password : 33333
Masukkan Username : rosihanari
Masukkan Password : 444
```

Maaf Anda hanya diberikan kesempatan maksimal 3 kali gagal login.

Sedangkan berikut ini tampilan jika login berhasil:

```
Masukkan Username : rosihanari
Masukkan Password : 123
```

Selamat datang 'rosihanari', Anda telah berhasil login.

## Functions

Selama ini kita sudah sering menggunakan perintah-perintah seperti `int()`, `str()`, `print()` dsb di dalam Python. Perintah-perintah tersebut secara spesifik disebut dengan 'function' atau fungsi. Di dalam Python, sebuah function bisa jadi sudah tersedia dan langsung bisa digunakan, seperti misalnya `int()`, `str()`, `print()` dll, tanpa harus kita mendefinisikan atau membuatnya terlebih dahulu. Function jenis ini dinamakan *built in functions*. Namun, ada juga function yang harus kita buat atau definisikan secara manual terlebih dahulu sebelum digunakan. Jika diperhatikan, ciri khas sebuah function adalah adanya tanda kurung `()` setelah nama functionnya, misalnya `print()`, `input()`, `str()` dll.

Gambaran sebuah function di dalam sebuah program adalah seperti subprogram di dalam sebuah program yang bertugas untuk menjalankan serangkaian perintah secara spesifik.

Analogi sebuah function dalam program adalah seperti sebuah organisasi. Di dalam organisasi tersebut terdapat beberapa divisi yang masing-masing divisi memiliki tugas tertentu secara spesifik. Setiap kali pimpinan organisasi akan menyelesaikan sebuah pekerjaan, maka cukup pimpinan organisasi meminta divisi tertentu yang sesuai dengan jenis pekerjaannya untuk menyelesaikannya. Dengan adanya keberadaan divisi dalam sebuah organisasi tentunya akan sangat membantu si pimpinan tanpa harus menyelesaikannya sendiri. Demikian pula di dalam sebuah program komputer. Seorang programmer bisa membuat sebuah subprogram (function) yang berfungsi sebagaimana divisi dalam organisasi tadi. Sehingga manfaat adanya function adalah bisa lebih efisien dalam proses coding dan terstruktur, serta terdapat konsistensi dalam alur program. Setiap kali programmer membutuhkan sebuah function yang

sudah dibuatnya untuk melakukan suatu proses, maka programmer cukup memanggil nama functionnya saja, tanpa harus koding lagi secara berulang-ulang dengan baris program yang sama.

## Cara Mendefinisikan Function

Sebuah function dibuat dengan tujuan melakukan tugas atau menjalankan serangkaian perintah tertentu, sesuai yang ditentukan.

Perhatikan contoh program berikut ini, di mana sebuah function kita buat atau definisikan.

```
def hello():
    print('Hallo')
    print('Hallo Bandung')
    print('Ibukota Priangan')

hello()
hello()
```

Untuk mendefinisikan sebuah function, digunakan perintah `def` kemudian diikuti dengan nama function yang akan dibuat, selanjutnya tambahkan tanda kurungnya. Dalam contoh tersebut, nama function yang dibuat adalah `hello()`.

Blok perintah

```
print('Hallo')
print('Hallo Bandung')
print('Ibukota Priangan')
```

adalah serangkaian perintah yang dijalankan ketika function `hello()` tersebut dipanggil.

Oleh karena itu ketika diberikan perintah

```
hello()
hello()
```

akan diperoleh output sebagai berikut:

```
Hallo
Hallo Bandung
Ibukota Priangan
Hallo
Hallo Bandung
Ibukota Priangan
```

dalam hal ini, function `hello()` dipanggil sebanyak dua kali.

Contoh function `hello()` yang diberikan tersebut merupakan function yang tidak memiliki parameter atau *argument*. Parameter dalam sebuah function berfungsi seperti halnya sebuah input bagi function tersebut supaya bisa bekerja atau berjalan dengan normal. Penulisan parameter sebuah function terletak di dalam tanda kurung setelah nama functionnya.

Perhatikan contoh ke dua berikut ini

```
def hello(nama):
    print('Hallo, selamat siang ' + nama)
    print('Nama Anda indah sekali')

hello('Rosihan Ari')
hello('Dwi Amalia')
hello('Faza Fauzan')
```

Pada contoh ke dua ini, function `hello()` memiliki sebuah parameter yaitu `nama`. Nilai dari parameter `nama` nantinya akan dicetak ketika function `hello()` ini dipanggil. Sehingga output dari program tersebut adalah

```
Hallo, selamat siang Rosihan Ari
Nama Anda indah sekali
Hallo, selamat siang Dwi Amalia
Nama Anda indah sekali
Hallo, selamat siang Faza Fauzan
Nama Anda indah sekali
```

Banyaknya parameter sebuah function bisa lebih dari satu sesuai kebutuhan yang diinginkan. Sebagai contoh, perhatikan contoh ke tiga berikut ini:

```
def penjumlahan(bil1, bil2):
    hasil = bil1 + bil2
    print(str(bil1) + ' + ' + str(bil2) + ' = ' + str(hasil))

penjumlahan(3, 5)
penjumlahan(2, -4)
```

Function `penjumlahan()` pada contoh di atas digunakan untuk menjumlahkan dua buah bilangan. Dalam hal ini function tersebut memiliki dua buah parameter yaitu `bil1` dan `bil2` sebagai kedua bilangan yang akan dijumlahkan (Catatan: antar parameter dipisahkan dengan tanda koma). Setelah dijumlahkan, selanjutnya hasil penjumlahannya akan ditampilkan. Adapun output dari program setelah dijalankan adalah

```
3 + 5 = 8
2 + -4 = -2
```

## Function dengan *Return Value*

Sebuah function dapat pula dibuat supaya menghasilkan suatu nilai (value) sebagai output dari function tersebut. Selanjutnya output atau value yang dihasilkan function tersebut dapat digunakan untuk proses yang lainnya.

Perhatikan contoh program berikut ini.

```
def penjumlahan(bil1, bil2):
    hasil = bil1 + bil2
    return hasil

a = 4
b = 3
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))

a = penjumlahan(2, 4)
b = penjumlahan(-1, 3)
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))

a = penjumlahan(penjumlahan(1, 2), 4)
b = penjumlahan(penjumlahan(-1, 4), penjumlahan(4, 5))
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))
```

Function `penjumlahan()` pada program di atas sedikit dimodifikasi dari contoh sebelumnya. Jika diperhatikan, maka di akhir bagian function diberikan perintah `return hasil`. Maksud dari perintah ini adalah membuat nilai dari variabel `hasil` sebagai output dari function `penjumlahan()`.

Sekarang kita lihat baris pada program berikut ini

```
a = 4
b = 3
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))
```

Baris program tersebut bermaksud untuk menjumlahkan dua bilangan `a` dan `b`, di mana masing-masing diberi nilai 4 dan 3. Di dalam perintah `print()` terdapat pemanggilan terhadap function `penjumlahan(a, b)` sehingga dalam hal ini function akan menjumlahkan bilangan 4 dan 3. Dari proses penjumlahan didapatkan `hasil = 7`. Nilai dari variabel `hasil` inilah yang menjadi output dan akan dikembalikan kepada perintah `penjumlahan(a, b)` tersebut. Sehingga perintah `penjumlahan(a, b)` bernilai 7. Akibatnya, akan dihasilkan tampilan

```
4 + 3 = 7
```

Selanjutnya pada baris:

```
a = penjumlahan(2, 4)
b = penjumlahan(-1, 3)
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))
```

Variabel `a` diberi nilai dari hasil penjumlahan 2 dan 4 yaitu 6, dan `b` diberi nilai hasil dari penjumlahan -1 dan 3 yaitu 2. Selanjutnya `print()` akan menampilkan hasil penjumlahan `a` dan `b` sebagai berikut

```
6 + 2 = 8
```

Sedangkan pada baris program:

```
a = penjumlahan(penjumlahan(1, 2), 4)
b = penjumlahan(penjumlahan(-1, 4), penjumlahan(4, 5))
print(str(a) + ' + ' + str(b) + ' = ' + str(penjumlahan(a, b)))
```

Nilai `a` diberi nilai dari hasil penjumlahan 3 dan 4, di mana 3 diperoleh dari hasil penjumlahan 1 dan 2. Sehingga dalam hal ini, nilai `a` bernilai 7. Sedangkan `b`, diberi nilai dari hasil penjumlahan 3 dan 9, di mana 3 diperoleh dari hasil penjumlahan -1 dan 4, dan 9 diperoleh dari hasil penjumlahan 4 dan 5. Oleh karena itu, nilai `b` adalah 12. Akibatnya output yang muncul adalah

```
7 + 12 = 19
```

## Lebih Lanjut dengan Function `print()`

Khusus untuk function `print()` yang sering kita gunakan, ada parameter tambahan bersifat optional yang bisa kita tambahkan. Selama ini, jika kita menggunakan `print()` maka outputnya akan menghasilkan baris baru di bawahnya. Sebagai contoh:

```
print('Hello')
print('World')
```

akan menghasilkan dua baris string ketika dijalankan

```
Hello
World
```

Hal tersebut disebabkan, ketika perintah `print()` diberikan secara default akhir dari string terdapat karakter `\n` (*newline*) secara implisit. Karakter *newline* ini dapat kita ubah dengan karakter apapun dengan cara menambahkan parameter `end='...'` di dalam `print()`, di mana titik-titiknya bisa diisi dengan sembarang karakter. Contoh:



```
print('Hello', end=' ')
print('World')
```

Di dalam contoh tersebut, akhir string 'Hello' ditambahkan karakter spasi kosong, sehingga dalam hal ini string 'World' yang dicetak setelahnya akan terletak di sebelah kanan 'Hello' didahului dengan karakter spasi kosong sebelumnya.

```
Hello World
```

Dengan demikian secara implisit, sebenarnya default dari `print()` terdapat parameter `end=' \n'`.

Oleh karena itu, penggunaan parameter `end='...'` pada `print()` dapat pula dimanfaatkan untuk keperluan input data di mana data yang dimasukkan oleh user berada di sebelah kanan string teks yang muncul. Contoh:

```
print('Masukkan sebuah kata : ', end=' ')
kata = input()
print('Kata yang Anda masukkan adalah : ' + kata)
```

Sehingga tampilan dari program di atas menjadi

```
Masukkan sebuah kata :  Hallo
Kata yang Anda masukkan adalah :  Hallo
```

yang dalam hal ini, posisi string yang diinputkan tidak terletak di bawah string 'Masukkan sebuah kata : '.

## Mengimport Function dari File Lain

Apabila sebuah function sudah dibuat/didefinisikan dan disimpan ke dalam sebuah file, maka kita bisa memanggil function tersebut di dalam file yang lainnya.

Sebagai contoh, misalkan diberikan sebuah file program Python dengan nama 'hello.py' yang di dalamnya terdapat sebuah function `hello1()` dan `hello2()` berikut ini

```
def hello1():
    print('Hello world')

def hello2():
    print('Hello everybody');
```

Selanjutnya misalkan kita punya file lain dengan '**coba.py**' yang di dalamnya terdapat perintah untuk memanggil function `hello1()` dan `hello2()` seperti di bawah ini

```
hello1()
hello2()
```

Apabila program `coba.py` dijalankan, maka akan muncul error sebagai berikut

```
Traceback (most recent call last):
  File "D:\coba.py", line 1, in <module>
    hello1()
NameError: name 'hello1' is not defined
```

Dari error tersebut, tampak bahwa kesalahan terjadi karena function `hello1()` tidak dikenal di dalam **coba.py**. Oleh karena itu, diperlukan cara bagaimana function-function yang ada di file **hello.py** bisa dipanggil di dalam **coba.py**. Adapun caranya adalah dengan memberikan perintah berikut ini sebelum memanggil nama functionnya:

```
from hello import *
```

Di dalam perintah tersebut, `hello` merupakan nama file yang berisi function-function yang akan dipanggil. Sedangkan tanda `*` bermakna bahwa yang akan dipanggil adalah semua function yang ada di dalam file tersebut. Sehingga isi dari file `coba.py` menjadi

```
from hello import *
hello1()
hello2()
```

Catatan: pastikan file berisi function yang akan diimpor terletak dalam direktori yang sama dengan file yang akan menggunakan function tersebut.

## Variabel Lokal dan Global

Pada sebuah program yang di dalamnya terdapat function, kita harus memahami penggunaan variabelnya. Dalam hal ini, kita harus memahami apa itu variabel lokal dan global. Untuk bisa memahaminya, perhatikan dua contoh program berikut ini.

### Program Pertama

```
def myFunction1():
    a = 4

print(a)
```

### Program Ke Dua

```
def myFunction2():
    print(a)

a = 4
myFunction2()
```

### Program Ke Tiga

```
def myFunction1():
    a = 4
def myFunction2():
    print(a)

myFunction1()
myFunction2()
```

Pada program pertama, terdapat function `myFunction1()` yang di dalamnya ada pendefinisian variabel `a` kemudian diberi nilai 4. Selanjutnya di luar function, terdapat perintah untuk mencetak nilai variabel `a`. Apabila program pertama dijalankan, maka akan terdapat error

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

Kesimpulan yang bisa diambil dari program pertama adalah ternyata variabel `a` yang didefinisikan di dalam function `myFunction1()` tidak dikenal di luar function tersebut.

Selanjutnya di dalam program ke dua, di luar function `myFunction2()` didefinisikan variabel `a` yang diberi nilai 4. Selanjutnya function `myFunction2()` dipanggil di mana akan mencetak nilai variabel `a` yaitu 4. Dalam hal ini, variabel `a` meskipun didefinisikan di luar function `myFunction2()` akan tetapi tetap dikenal di dalam function tersebut.

Sedangkan pada program ke tiga, ketika dijalankan maka juga akan menghasilkan error

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

Adapun penyebab error tersebut adalah bahwa variabel `a` yang didefinisikan di dalam function `myFunction1()` tidak dikenal di dalam `myFunction2()`.

Berdasarkan ketiga contoh program yang telah diberikan, dapat disimpulkan bahwa variabel yang didefinisikan di dalam sebuah function adalah bersifat lokal yaitu hanya bisa dikenal dan digunakan di dalam function itu sendiri. Hal ini dapat dilihat pada program pertama dan ke tiga. Sedangkan variabel yang didefinisikan di program utama (bukan di dalam function) maka juga akan dikenali di dalam function apapun. Hal ini dapat dilihat pada program ke dua. Sehingga dalam hal ini sifat variabel tersebut adalah global atau dikenali di semua bagian dalam program.

Selanjutnya, akan diberikan sebuah contoh program berikut ini:

```
def myFunction1():
    a = 4

a = 10
myFunction1()
print(a)
```

Pada program tersebut, mula-mula variabel `a` diberi nilai 10 di dalam program utama. Selanjutnya terjadi pemanggilan function `myFunction1()` di mana pada function ini ada proses assignment terhadap variabel `a` dengan nilai 4. Setelah itu, nilai variabel `a` dicetak dan akan memberikan hasil 10. Mengapa nilai variabel `a` nya 10, bukan 4? Apabila program tersebut diperhatikan, maka ada dua pendefinisian variabel `a`, yaitu di program utama dan di dalam function `myFunction1()`. Meskipun nama variabelnya sama, akan tetapi keduanya berbeda yang disebabkan sifat dari keduanya. Variabel `a` yang didefinisikan di dalam program utama adalah bersifat global. Sebaliknya, variabel `a` yang didefinisikan di dalam function `myFunction1()` bersifat lokal. Oleh karena itu, ketika perintah `print(a)` di dalam program utama dijalankan maka variabel `a` di sini adalah variabel `a` yang bersifat global.

Pertanyaan berikutnya adalah bagaimana caranya supaya nilai `a` yang awalnya diberi nilai 10 pada program di atas, yang dalam hal ini sebagai variabel global bisa diubah nilainya menjadi 4 di dalam function `myFunction1()`? Untuk melakukan hal ini caranya adalah dengan memberikan label 'global' di pada variabel `a` di dalam `myFunction1()` yang bertujuan untuk memberi tanda bahwa variabel `a` yang disebutkan di dalam function tersebut adalah variabel `a` yang bersifat global, sehingga menjadi

```
def myFunction1():
    global a
    a = 4

a = 10
myFunction1()
print(a)
```

Setelah perubahan pada program akan dihasilkan output 4, yang dalam hal ini merupakan nilai variabel `a` yang sudah diubah nilainya.

## Penanganan Exception (*Exception Handling*)

Suatu program yang dibuat terkadang bisa berhenti secara abnormal (*crash*) yang disebabkan oleh adanya ketidaksesuaian tipe data baik ketika proses input atau output, atau bisa juga disebabkan proses perhitungan yang tidak semestinya, misalnya pembagian dengan nol.

Perhatikan contoh function berikut ini:

```
def pembagian(a, b):
    hasil = a/b
    print('Hasilnya : ' + str(hasil))

pembagian(3, -5)
pembagian(10, 2.5)
pembagian(9, 0)
pembagian(-2, -4)
pembagian(1, 2)
```

Di dalam program di atas terdapat function `pembagian(a, b)` yang menampilkan hasil pembagian `a/b`.

Apabila program di atas dijalankan, maka akan dihasilkan tampilan berikut ini

```
Hasilnya : -0.6
Hasilnya : 4.0
Traceback (most recent call last):
  File "D:\exception.py", line 7, in <module>
    pembagian(9, 0)
  File "D:\exception.py", line 2, in pembagian
    hasil = a/b
ZeroDivisionError: division by zero
```

Berdasarkan tampilan tersebut, dapat kita lihat bahwa program berhenti (*crash*) dengan pesan `ZeroDivisionError: division by zero` ketika proses pemanggilan `pembagian(9, 0)`. Hal ini disebabkan adanya proses pembagian dengan nol.

Untuk mengantisipasi program crash ini atau supaya program tetap berjalan sampai dengan selesai, kita bisa melakukan *exception handling*. Proses exception handling dilakukan dengan membuat blok dengan sintaks sebagai berikut:

```
try:
    ...
except [errorMsg]:
    ...
```

Blok `try – except` di atas digunakan untuk menangkap pesan error (*error message*) yang muncul karena crashnya program. Setelah pesan error ditangkap selanjutnya diganti dengan proses yang lainnya.

Sehingga apabila `try-except` ini diterapkan pada contoh program yang diberikan di awal pembahasan maka akan menjadi

```
def pembagian(a, b):
    try:
        hasil = a/b
        print('Hasilnya : ' + str(hasil))
    except ZeroDivisionError:
        print('Pembagian dengan nol')

pembagian(3, -5)
pembagian(10, 2.5)
pembagian(9, 0)
pembagian(-2, -4)
pembagian(1, 2)
```

Setelah penggunaan try-except maka akan muncul hasil sebagai berikut:

```
Hasilnya : -0.6
Hasilnya : 4.0
Pembagian dengan nol
Hasilnya : 0.5
Hasilnya : 0.5
```

Penggunaan try-except di atas diterapkan pada sebuah function. Namun, bisa juga try-except diterapkan ketika proses pemanggilan functionnya seperti contoh berikut ini:

```
def pembagian(a, b):
    hasil = a/b
    print('Hasilnya : ' + str(hasil))

try:
    pembagian(3, -5)
    pembagian(10, 2.5)
    pembagian(9, 0)
    pembagian(-2, -4)
    pembagian(1, 2)
except ZeroDivisionError:
    print('Pembagian dengan nol')
```

Apabila program di atas dijalankan maka akan dihasilkan:

```
Hasilnya : -0.6
Hasilnya : 4.0
Pembagian dengan nol
```

Terdapat perbedaan tampilan dibandingkan dengan program sebelumnya, yaitu dalam hal ini pembagian(-2, -4) dan pembagian(1, 2) tidak dijalankan karena begitu terjadi exception

yaitu ketika menjalankan pembagian(9, 0) program tidak bisa kembali melanjutkan proses yang ada di bagian try.

### Studi Kasus

Buatlah program permainan game sederhana yaitu 'tebak angka'. Ketika program dijalankan, program akan mengacak sebuah angka bilangan bulat antara 1 s/d 30. Selanjutnya program meminta user menebak angka tersebut dengan memasukkan sebuah angka. Jika angka yang ditebak si user lebih dari angka yang dipilih komputer maka program akan menampilkan pesan 'Angka tebakan Anda terlalu tinggi'. Sedangkan jika tebakan angkanya terlalu rendah maka muncul pesan 'Angka tebakan Anda terlalu rendah'. Gunakan function untuk meminta user menebak angkanya serta menampilkan pesan salah atau tidak tebakan tersebut.

Berikut ini contoh tampilannya:

```
Hallo, saya telah memikirkan sebuah angka antara 1 s/d 25. Bisakah
Anda tebak berapa?

Coba tebak berapa: 10
Hehe... maaf salah, tebakan Anda terlalu tinggi
Coba tebak berapa: 7
Hehe... maaf salah, tebakan Anda terlalu rendah
Coba tebak berapa: 8
Hehe... maaf salah, tebakan Anda terlalu rendah
Coba tebak berapa: 9
Yes!!! Anda tepat sekali. Selamat !!!
```

### Petunjuk:

Gunakan perintah `random.randint(a, b)` untuk mengacak bilangan bulat mulai dari a s/d b. Untuk bisa menggunakan perintah ini, lakukan import pada modul random. Contoh:

```
import random

bilAcak = random.randint(1, 4)
print('Bil Acak 1 s.d 4 adalah : ' + str(bilAcak))
```

### Jawab:

```
def mintaMenebak():
    try:
        print('Coba tebak berapa : ', end='')

        # melabeli global pada var tebak
        global tebak
        # mengubah nilai var tebak berdasarkan input
        tebak = int(input())
```

```

        if tebak > bil:
            print('Hehe... maaf salah, tebakkan Anda terlalu tinggi')
        elif tebak < bil:
            print('Hehe... maaf salah, tebakkan Anda terlalu rendah')
        else:
            print('Yes!!! Anda tepat sekali. Selamat !!!')
    except ValueError:
        print('Invalid input')

# mengimport modul random
import random

# merandom bilangan bulat 1 s/d 30
bil = random.randint(1, 30)

print('Hallo, saya telah memilih sebuah angka antara 1 s/d 30.')
print('Bisakah Anda menebak angka berapa itu?')

# nilai tebak mula-mula
tebak = 0

# perulangan untuk meminta user menebak
while True:
    mintaMenebak()
    if tebak == bil:
        break

```

Di dalam program utama pada program di atas, terdapat perulangan while yang digunakan untuk mengulang permintaan menebak kepada pengguna. Perulangan baru berhenti jika bilangan yang ditebak (variabel `tebak`) sama dengan bilangan acak yang dipilih oleh komputer (`bil`). Dalam hal ini variabel `tebak` bersifat global, demikian pula variabel `bil`. Selanjutnya untuk menampilkan permintaan user supaya menebak bilangan menggunakan function `mintaMenebak()`. Di dalam function `mintaMenebak()` terdapat deklarasi global `tebak` yang digunakan untuk memberi label bahwa variabel `tebak` yang digunakan dalam function tersebut adalah variabel `tebak` yang bersifat global yang nilai awalnya adalah 0. Dengan demikian, nilai variabel `tebak` ini akan berubah-ubah sesuai input bilangan yang dimasukkan oleh user. Jika di dalam function `mintaMenebak()` tidak diberi label global pada variabel `tebak`, maka dianggap variabel lokal saja, dan akibatnya perulangan while tidak akan pernah berhenti karena nilai variabel `tebak` yang global tidak akan pernah berubah yaitu nol.

Selain itu, di dalam function `mintaMenebak()` juga diberikan validasi input bilangan menggunakan try-except untuk mencegah program crash yang disebabkan input yang salah. Dalam hal ini jenis error yang dibaca adalah `ValueError`.

Output dari program di atas adalah:



```

Hallo, saya telah memilih sebuah angka antara 1 s/d 30.
Bisakah Anda menebak angka berapa itu?
Coba tebak berapa : 4
Hehe... maaf salah, tebakkan Anda terlalu tinggi
Coba tebak berapa : a
Invalid input
Coba tebak berapa : 2
Hehe... maaf salah, tebakkan Anda terlalu rendah
Coba tebak berapa : 3
Yes!!! Anda tepat sekali. Selamat !!!

```



## Soal Latihan

1. Buatlah program Python untuk menentukan apakah pasangan tiga nilai bilangan bulat positif  $a$ ,  $b$ ,  $c$  merupakan triple pitagoras atau bukan. Gunakan function `triplePitagoras(a, b, c)` untuk menentukan hal ini.  
Keterangan: function harus bisa handle apabila bilangan nilai parameternya bukan bilangan bulat positif
2. Buatlah program Python untuk menentukan bilangan terbesar dari tiga nilai bilangan  $a$ ,  $b$ ,  $c$ .  
Gunakan function `maksimum(a, b, c)` untuk menentukan bilangan terbesar tersebut.
3. Buatlah function `deret(a, b, n)` yang berfungsi untuk menampilkan deret aritmatika dengan suku awal  $a$ , beda (selisih)  $b$ , sebanyak  $n$  suku ( $n$  adalah bilangan bulat positif).  
Keterangan: function harus bisa handle apabila bilangan  $n$  nya bukan bilangan bulat positif
4. Buatlah function `kotak(n)` yang berfungsi untuk membuat formasi kotak, dengan  $n$  adalah bilangan bulat positif.

Misal  $n = 3$ , maka formasi kotak yang terbentuk adalah:

```

XXX
XXX
XXX
XX
XX
X

```

Misal  $n = 4$ , maka formasi kotak yang terbentuk adalah:

```

XXXX
XXXX
XXXX
XXX
XXX
XXX
XX
XX
X

```

Keterangan:

function harus bisa handle apabila bilangan  $n$  nya bukan bilangan bulat positif

5. Buatlah function `kabisat(year)` untuk menentukan apakah tahun 'year' merupakan tahun kabisat atau bukan.

Keterangan:

- Gunakan aturan penentuan tahun kabisat dari [https://id.wikipedia.org/wiki/Tahun\\_kabisat](https://id.wikipedia.org/wiki/Tahun_kabisat)
- Function harus bisa handle tahun yang bukan bilangan bulat positif

6. Buatlah function dengan nama `collatz(bil)`, di mana function ini menghasilkan nilai `bil` // 2 jika `bil` nya merupakan bilangan genap, dan menghasilkan nilai  $3 * bil + 1$  jika `bil` nya bilangan ganjil. Selanjutnya buat program yang meminta user untuk memasukkan sembarang bilangan bulat positif. Kemudian panggil function `collatz()` terhadap bilangan tersebut dan panggil kembali `collatz()` terhadap hasil tersebut. Demikian proses ini diulang hingga hasil akhir nya diperoleh nilai 1.

Contoh:

Masukkan bilangan bulat positif : 3

Output:

```
3
10
5
16
8
4
2
1
```

## List

List di dalam Python merupakan suatu nilai yang berisi beberapa nilai yang tersaji secara terurut. Nilai-nilai yang terdapat dalam list tidak harus bertipe data sama. List secara struktur mirip dengan tipe data array di dalam bahasa pemrograman lainnya. Namun perbedaannya adalah jika array nilai yang tersimpan di dalamnya harus bertipe data yang sama, misalnya array adalah bahasa Pascal, C/C++.

List ini memiliki kelebihan dibanding variabel tunggal yang biasa, yaitu sifat fleksibilitasnya sehingga kita bisa menyimpan banyak nilai sekaligus ke dalam sebuah variabel saja.

Berikut ini contoh cara mendefinisikan suatu nilai dalam bentuk list:

```
>>> buah = ['mangga', 'jeruk', 'apel', 'anggur']
>>> buah
['mangga', 'jeruk', 'apel', 'anggur']
```

Dalam contoh di atas, semua nilai dalam list adalah berupa string. Sedangkan pada contoh berikut ini di dalam list terdapat nilai dengan beberapa jenis tipe data yang berbeda.

```
>>> myList = [1, 'hallo', 'apa kabar', 0.001]
>>> myList
[1, 'hallo', 'apa kabar', 0.001]
```

### Membaca Sebuah Nilai dalam List

Untuk membaca sebuah nilai dalam sebuah list dengan cara menyebutkan nomor urut nilai tersebut dalam listnya. Contoh:

```
>>> buah = ['mangga', 'jeruk', 'apel', 'anggur']
>>> buah[0]
'mangga'
>>> buah[1]
'jeruk'
>>> buah[2]
'apel'
>>> buah[3]
'anggur'
```

Dari contoh di atas, dapat disimpulkan bahwa nomor urut nilai dalam list dimulai dari nilai ke-0. Nilai berikutnya adalah nilai ke-1, 2 dst. Nomor urut ini selanjutnya dinamakan indeks. Sehingga jika suatu list terdapat n nilai, maka nomor indeksinya dimulai dari indeks ke-0, 1, 2, ..., n-1 (indeks harus berupa bilangan bulat).

Berikut ini contoh pembacaan nilai dalam list kemudian digunakan untuk operasi aritmatika.

```
>>> data = [2, 1, 'hello', -3, 'python']
>>> data[0] + 2
4
>>> data[0] + data[3]
-1
>>> data[2] + ' apa kabar? ' + 'I love ' + data[4]
'hello apa kabar? I love python'
>>> data[0] + 'hello world'
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    data[0] + 'hello world'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Pada operasi `data[0] + 2` bermakna bahwa nilai pada indeks ke-0 dari list data dijumlahkan dengan 2, sehingga dalam hal ini `data[0] + 2 = 2 + 2 = 4`. Operasi `data[0] + data[3]` menjumlahkan nilai indeks ke-0

dan indeks ke-3, sehingga  $\text{data}[0] + \text{data}[3] = 2 + (-3) = -1$ . Sedangkan pada operasi  $\text{data}[2] + \text{'apa kabar? ' + 'I love ' + data}[4]$  merupakan operasi penggabungan string (string concatenation) terhadap nilai indeks ke-2 dari list data, string 'apa kabar?', 'I love' dan string dari indeks ke-4 dari list data. Sehingga  $\text{data}[2] + \text{'apa kabar? ' + 'I love ' + data}[4] = \text{'hello' + ' apa kabar? ' + 'I love' + 'python'} = \text{'hello apa kabar? I love python'}$ . Selanjutnya pada operasi  $\text{data}[0] + \text{'hello world'}$  terdapat error karena kedua operand, dalam hal ini  $\text{data}[0]$  dan string 'hello world' tidak bertipe data yang sama, yaitu bertipe data bilangan dan yang satunya adalah string sehingga tidak bisa dilakukan operasi +.

Nilai di dalam sebuah list juga bisa merupakan list juga, berikut ini contohnya:

```
>>> matriks = [[1, 2], [3, 4], [5, 6, 7], 8, 9]
>>> matriks[0]
[1, 2]
>>> matriks[0] + matriks[1]
[1, 2, 3, 4]
>>> matriks[0][0]
1
>>> matriks[2][1]
6
```

Perintah  $\text{matriks}[0]$  bermakna mengambil nilai indeks ke-0 dari list matriks, dalam hal ini nilainya berupa list juga yaitu  $[1, 2]$ . Perintah  $\text{matriks}[0] + \text{matriks}[1]$  merupakan operasi penggabungan dua buah list, dalam hal ini  $[1, 2]$  digabung dengan  $[3, 4]$  sehingga menghasilkan  $[1, 2, 3, 4]$ . Perintah  $\text{matriks}[0][0]$  adalah digunakan untuk membaca nilai indeks ke-0 dari list  $\text{matriks}[0]$  yang dalam hal ini  $[1, 2]$  sehingga diperoleh 1. Demikian pula perintah  $\text{matriks}[2][1]$  yaitu membaca nilai indeks ke-1 dari list  $\text{matriks}[2]$  dalam hal ini  $[5, 6, 7]$  yaitu 6.

## Indeks List Negatif

Dalam contoh-contoh sebelumnya, penomoran indeks suatu list dimulai dari indeks ke-0, 1,... dst.

Namun bisa juga kita menggunakan nomor indeks negatif untuk membaca suatu nilai dalam list.

Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga']
>>> buah[-1]
'mangga'
>>> buah[-2]
'jeruk'
>>> buah[-3]
'apel'
>>> buah[-2] + ' minum ' + buah[-2]
'jeruk minum jeruk'
```

Dari contoh di atas, bisa disimpulkan bahwa nomor indeks (-1) adalah mengacu pada nilai terakhir dari list. Nomor indeks (-2) adalah sebelah kirinya indeks (-1), demikian seterusnya. Dengan begitu andaikan

ada suatu list dengan n nilai, maka penomoran indeksinya jika dimulai dari paling terakhir adalah -1, -2, -3, ..., -n.

### Membaca Sublist dari List

Sebuah sublist dapat pula dibaca nilainya dari sebuah list. Perhatikan contoh berikut ini:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah[0:3]
['apel', 'jeruk', 'mangga']
>>> buah[1:5]
['jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah[1:]
['jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah[:4]
['apel', 'jeruk', 'mangga', 'durian']
```

Perintah buah[0:3] bermakna bahwa nilai list yang diambil adalah mulai dari indeks ke-0 sampai dengan indeks ke-3 dari list buah, namun tidak termasuk nilai indeks ke-3 nya. Demikian pula perintah buah[1:5] bermakna bahwa nilai list yang diambil adalah mulai dari indeks ke-1 sampai indeks ke-5, namun tidak termasuk nilai indeks ke-5 nya.

Cara lain membaca sublist adalah dengan cara seperti pada contoh yaitu buah[1:]. Perintah tersebut bermakna membaca nilai mulai pada indeks ke-1 sampai dengan akhir list. Demikian juga perintah buah[:5] yaitu membaca nilai mulai dari awal indeks sampai dengan indeks ke-5, tidak termasuk nilai pada indeks ke-5 nya tersebut.

### Fungsi len( ) pada List

Di dalam Python, terdapat function khusus untuk mengetahui jumlah nilai atau ukuran dari suatu list yaitu function len( ). Contoh:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> len(buah)
5
```

### Mengubah Nilai pada Indeks Tertentu dalam List

Suatu nilai pada indeks tertentu dalam list dapat diubah nilainya. Berikut ini contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah[1] = 'anggur'
>>> buah
['apel', 'anggur', 'mangga', 'durian', 'klengkeng']
>>> buah[-2] = 'melon'
>>> buah
['apel', 'anggur', 'mangga', 'melon', 'klengkeng']
>>> buah[2] = buah[1] + ' dan ' + buah[4]
```

```
>>> buah
['apel', 'anggur', 'anggur dan klengkeng', 'melon', 'klengkeng']
```

Pada contoh di atas, perintah `buah[1] = 'anggur'` bermakna mengganti nilai pada indeks ke-1 pada list buah dengan string 'anggur'. Sehingga dalam hal ini, nilai yang sebelumnya adalah 'jeruk' diganti menjadi 'anggur'. Demikian pula yang terjadi pada perintah `buah[-2] = 'melon'` yaitu mengganti nilai 'durian' menjadi 'melon'. Sedangkan `buah[2] = buah[1] + ' dan ' + buah[4]` mengganti nilai 'mangga' dengan string 'anggur dan klengkeng'.

### Penggabungan List (*List Concatenation*) dan Replikasinya

Sebuah list dapat digabung dengan list lainnya seperti halnya string, atau hal ini diistilahkan dengan list concatenation. Adapun cara menggabungkan list ini menggunakan operator `+`. Contoh:

```
>>> list1 = [2, 3, 4]
>>> list2 = ['amir', 'budi', 'cici']
>>> list1 + list2
[2, 3, 4, 'amir', 'budi', 'cici']
```

Sedangkan untuk mereplikasi sebuah list menggunakan operator `*`. Contoh:

```
>>> list1 = [1, 2, 3]
>>> list1 * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

### Menghapus Nilai dalam List

Sebuah nilai pada indeks tertentu dalam suatu list dapat dihapus dengan memberikan perintah `del`. Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> del buah[1]
>>> buah
['apel', 'mangga', 'durian', 'klengkeng']
```

Perintah `del buah[1]` pada contoh tersebut bermakna menghapus nilai pada indeks ke-1 dari list buah, dalam hal ini adalah 'jeruk'.

Catatan:

Perintah `del` juga dapat digunakan untuk menghapus variabel yang sudah tidak diperlukan lagi dalam Python.

## Bekerja dengan List

Setelah kita mengetahui apa itu list, cara mendefinisikannya, serta operasinya, selanjutnya akan dibahas bagaimana cara bekerja dengan list.

### Input Data List

Pembahasan pertama tentang subbab 'Bekerja dengan List' adalah tentang bagaimana caranya menyimpan data input program ke dalam list. Perhatikan contoh berikut ini:

```
# Program yang membaca input beberapa data
# berupa bilangan bulat, kemudian menampilkan rata-ratanya.

# Input diakhiri dengan memasukkan nilai 0

dataNilai = []
i = 1
while True:
    print('Masukkan data ke-' + str(i) + ' : ', end='')
    x = int(input())
    if x == 0:
        break
    else:
        dataNilai = dataNilai + [x]
    i = i + 1

# proses perhitungan rata-rata

sum = 0
for i in range(len(dataNilai)):
    sum = sum + dataNilai[i]

rerata = sum/len(dataNilai)

# output
print('Rata-ratanya : ' + str(rerata))
```

Program di atas untuk membaca input serangkaian bilangan bulat. Selanjutnya outputnya adalah rata-rata dari serangkaian bilangan bulat yang diinputkan tersebut. Dalam hal ini, sebagai penanda akhir dari proses input adalah dengan memasukkan bilangan 0.

Mula-mula, kita definisikan sebuah list untuk menyimpan data nilai bilangan yang akan diinputkan, yaitu list bernama `dataNilai`. Awalnya diberi nilai list kosong karena belum ada nilainya menggunakan perintah `dataNilai = []`. Selanjutnya dengan menggunakan looping `while`, proses input dilakukan beberapa kali. Untuk memasukkan bilangan yang diinputkan ke dalam list, gunakan perintah `dataNilai = dataNilai + [x]` yang dalam hal ini `x` adalah bilangan yang diinputkan.

Output tampilan dari program tersebut jika dijalankan adalah:

```
Masukkan data ke-1 : 3
Masukkan data ke-2 : 4
Masukkan data ke-3 : 5
Masukkan data ke-4 : 0
Rata-ratanya : 4.0
```

### Looping dalam List

Perhatikan kembali pada program menghitung rata-rata sebelumnya. Selanjutnya untuk menghitung rata-rata, berarti langkah awal kita harus jumlahkan seluruh nilai yang ada dalam list `dataNilai`. Pada program di atas langkah ini dilakukan dengan menggunakan looping `for`. Looping `for` ini diulang sebanyak `n` kali di mana `n` adalah adalah banyaknya nilai dalam `dataNilai` (nilai `n` bisa didapat dengan perintah `len(dataNilai)`). Dalam hal ini, looping `for` dinyatakan dalam bentuk

```
for i in range(len(dataNilai)):
    sum = sum + dataNilai[i]
```

Variabel `i` nantinya digunakan sebagai nomor indeks untuk membaca nilai pada setiap indeks dari list `dataNilai`, di mana `i` nya bernilai `0, 1, 2, ..., len(dataNilai) - 1`.

Selain dengan cara looping di atas, dapat pula looping dilakukan dengan cara sebagai berikut:

```
# Program yang membaca input beberapa data
# berupa bilangan bulat, kemudian menampilkan rata-ratanya.

# Input diakhiri dengan memasukkan nilai 0

dataNilai = []
i = 1
while True:
    print('Masukkan data ke-' + str(i) + ' : ', end='')
    x = int(input())
    if x == 0:
        break
    else:
        dataNilai = dataNilai + [x]
    i = i + 1

sum = 0

for nilai in dataNilai:
    sum = sum + nilai

rerata = sum/len(dataNilai)
print('Rata-ratanya : ' + str(rerata))
```

Perhatikan pada bagian looping:



```
for nilai in dataNilai:
    sum = sum + nilai
```

Pada looping tersebut, kita tanpa menyebutkan range indeksnya, dan tanpa perlu tahu banyaknya nilai dalam list. Secara otomatis looping akan dilakukan sebanyak nilai dalam list `dataNilai`. Variabel `nilai` di situ digunakan untuk membaca setiap nilai dari list `dataNilai` pada setiap indeksnya.

### Operator `in` dan `not in`

Untuk mengecek keberadaan suatu nilai dalam suatu list, dapat dilakukan menggunakan operator `in` atau `not in`. Kedua operator ini menghasilkan nilai boolean (`True` atau `False`). Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> 'pepaya' in buah
False
>>> 'jeruk' in buah
True
>>> 'durian' not in buah
False
>>> 'jambu' not in buah
True
```

Berdasarkan contoh di atas, operator `in` digunakan untuk mengecek apakah suatu nilai terdapat dalam suatu list atau tidak, jika ya maka menghasilkan `True`, sedangkan jika tidak maka bernilai `False`. Sebaliknya operator `not in` digunakan untuk mengecek apakah suatu nilai tidak terdapat dalam suatu list atau tidak. Jika ya (tidak terdapat dalam list) maka akan menghasilkan `True`, sedangkan jika tidak (berarti ada dalam list) menghasilkan nilai `False`.

### Mencari Nilai Tertentu dalam List dengan `index()`

Di dalam Python, dapat dengan mudah kita mencari suatu nilai tertentu dalam sebuah list. Kita dapat dengan mudah mengetahui indeks ke berapa suatu nilai itu berada dalam list. Untuk mengetahui indeks ke berapa dalam sebuah list yang berisi nilai tertentu menggunakan `index()`. Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah.index('mangga')
2
```

### Menambahkan Nilai pada List dengan `append()` dan `insert()`

Untuk menambahkan nilai ke dalam suatu list dapat pula dilakukan dengan `append()` dan `insert()`. Perbedaan keduanya adalah bahwa jika `append()` penambahannya dilakukan di akhir list, sedangkan `insert()` penambahannya ke dalam nomor indeks tertentu. Contoh penggunaan `append()`:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah.append('manggis')
```

```
>>> buah
['apel', 'jeruk', 'mangga', 'durian', 'klengkeng', 'manggis']
```

Sedangkan contoh penggunaan `insert()` adalah:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah.insert(2, 'pepaya')
>>> buah
['apel', 'jeruk', 'pepaya', 'mangga', 'durian', 'klengkeng']
```

Pada contoh di atas, proses penambahan nilai 'pepaya' dilakukan pada indeks ke-2 dari list buah. Sehingga posisi dari nilai 'mangga', 'durian', dan 'klengkeng' bergeser yang tadi masing-masing di posisi indeks ke-2, 3, dan 4 menjadi indeks ke-3, 4, dan 5.

### Menghapus Nilai Tertentu dalam List dengan `remove()`

Jika sebelumnya sudah dibahas tentang penggunaan perintah `del` untuk menghapus suatu nilai pada indeks tertentu dari sebuah list, maka nilai dalam suatu list juga dapat dihapus dengan perintah `remove()`. Perbedaannya adalah bahwa untuk `del` kita harus mengetahui nomor indeks mana yang akan dihapus, sedangkan `remove()` yang perlu diketahui hanyalah nilainya saja. Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng', 'apel']
>>> buah.remove('apel')
>>> buah
['jeruk', 'mangga', 'durian', 'klengkeng']
```

Contoh di atas menunjukkan proses penghapusan nilai 'apel' dalam list buah.

### Mengurutkan Nilai dalam List dengan `sort()`

Pengurutan nilai dalam sebuah list juga dapat dilakukan, yaitu dengan menggunakan `sort()`. Secara default `sort()` akan mengurutkan secara *ascending*. Contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah.sort()
>>> buah
['apel', 'durian', 'jeruk', 'klengkeng', 'mangga']
```

Namun, dapat pula pengurutannya dilakukan secara *descending*, contohnya:

```
>>> buah = ['apel', 'jeruk', 'mangga', 'durian', 'klengkeng']
>>> buah.sort(reverse=True)
>>> buah
['mangga', 'klengkeng', 'jeruk', 'durian', 'apel']
```

Untuk pengurutan secara *descending*, perlu penambahan parameter `reverse=True` pada `sort()`.

## Tipe Data yang Mirip dengan List

Seperti yang sudah kita pahami bahwa suatu list terdapat beberapa nilai yang tersimpan di dalamnya. Adapun untuk mengakses nilai tersebut, digunakan nomor indeks. Selain list, ternyata ada juga tipe data lain yang mirip dengan list, yaitu tipe data string dan tuple.

### String

Suatu string dapat kita perlakukan mirip dengan list. Contohnya:

```
>>> myName = 'Rosihan Ari'
>>> myName[0]
'R'
>>> myName[4]
'h'
```

Dalam contoh di atas, dapat kita lihat bahwa kita bisa mengakses atau membaca karakter penyusun string tersebut dengan cara yang sama seperti list. Meskipun mirip, namun karakteristik string dan list adalah berbeda. Dalam hal ini string bersifat *immutable* (kekal) sedangkan list bersifat *mutable* (dapat berubah). Apa maksudnya?

Seperti yang telah kita bahas sebelumnya tentang list, bahwa suatu list dapat kita tambahkan nilai ke dalamnya, atau kita hapus nilai-nilai di dalamnya, atau dilakukan sorting dsb, yang artinya bahwa list strukturnya bisa berubah melalui serangkaian operasi. Sehingga dalam hal ini list bersifat *mutable*.

Namun sebaliknya, suatu string jika sudah didefinisikan nilainya maka dia bersifat *immutable*. Sehingga kita tidak bisa mengubah struktur penyusun stringnya. Contohnya:

```
>>> myName = 'Rosihan Ari'
>>> myName[0] = 'S'
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    myName[0] = 'S'
TypeError: 'str' object does not support item assignment
>>> del myName[0]
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <module>
    del myName[0]
TypeError: 'str' object doesn't support item deletion
```

Pada contoh di atas, kita tidak bisa mengubah atau menghapus nilai indeks ke-0 dari string `myName`. Sehingga akibatnya muncul error.

Dengan demikian, yang bisa dilakukan terhadap tipe data *immutable* seperti string ini adalah hanyalah mengakses/membaca nilai-nilainya saja sebagaimana list. Contohnya:

```
>>> myName = 'Rosihan Ari'
```

```
>>> myName[1]
'o'
>>> myName[0:5]
'Rosih'
>>> myName[:-2]
'Rosihan A'
>>> myName + ' Yuana'
'Rosihan Ari Yuana'
```

## Tuple

Tipe data tuple ini sangat mirip dengan list. Perbedaannya hanyalah pada jenis kurung yang digunakannya, serta tuple bersifat *immutable* seperti halnya string. Contohnya:

```
>>> myTuple = (1, 2, 'hello')
>>> myTuple[2]
'hello'
>>> myTuple[0] + myTuple[1]
3
>>> myTuple[2] = 'hello world'
Traceback (most recent call last):
  File "<pyshell#100>", line 1, in <module>
    myTuple[2] = 'hello world'
TypeError: 'tuple' object does not support item assignment
```

Dari contoh di atas, dapat diketahui bahwa nilai dalam sebuah tuple hanya bisa diakses saja dengan cara yang sama seperti list. Akan tetapi, dikarenakan tuple bersifat *immutable* maka struktur tuple tidak bisa diubah.

## Mengkonversi ke Tuple dan List

Setelah kita tahu sifat dari tuple dan list, maka di dalam Python terdapat perintah untuk bisa mengkonversi tipe data tuple ke list, dan juga sebaliknya. Perintah untuk mengubah suatu list ke tuple dengan cara memberikan perintah `tuple()`, dan untuk mengubah tuple ke list dengan perintah `list()`. Contohnya:

```
>>> myList = [1, 2, 3, 4]
>>> myTuple = (1, 2, 3, 4)
>>> myString = 'I Love Python'
>>> tuple(myList)
(1, 2, 3, 4)
>>> list(myTuple)
[1, 2, 3, 4]
>>> list(myString)
['I', ' ', 'L', 'o', 'v', 'e', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

### Mengcopy List

Maksud dari mengcopy di sini adalah menyalin nilai pada suatu variabel list ke variabel lainnya. Tidak seperti mengcopy nilai variabel yang tunggal, di dalam Python proses mengcopy list tidak sesederhana itu. Jika untuk variabel tunggal, maka cukup dengan memberikan perintah assignment seperti pada contoh berikut ini:

```
>>> a = 10
>>> b = a
>>> b
10
```

Namun hal tersebut tidak berlaku untuk list. Contohnya:

```
>>> list1 = ['a', 'b', 'c']
>>> list2 = list1
>>> list2
['a', 'b', 'c']
>>> list2[0] = 'd'
>>> list2
['d', 'b', 'c']
>>> list1
['d', 'b', 'c']
```

Jika kita perhatikan contoh di atas, maka `list2` memiliki nilai yang sama dengan `list1`. Namun jika indeks ke-0 dari `list2` ini diubah menjadi `'d'`, maka indeks ke-0 dari `list1` juga ikut berubah. Demikian juga seandainya kita lakukan perubahan nilai pada salah satu indeks di `list1`, maka di dalam `list2` juga ikut terjadi perubahan.

Selanjutnya, bagaimana supaya menghindari hal tersebut di atas sehingga kita bisa mengcopy dua atau lebih list yang nilainya sama namun saling independen? Caranya adalah dengan menggunakan perintah `copy()`. Contoh:

```
>>> import copy
>>> list1 = ['a', 'b', 'c']
>>> list2 = copy.copy(list1)
>>> list2
['a', 'b', 'c']
>>> list1[0] = 'd'
>>> list1
['d', 'b', 'c']
>>> list2
['a', 'b', 'c']
```

Berdasarkan contoh di atas, sebelum menjalankan perintah `copy()` terlebih dahulu harus mengimport modul bernama `'copy'`. Apabila kita perhatikan, maka pada contoh tersebut dapat diperoleh dua buah list yaitu `list1` dan `list2` yang saling independen.

### Studi Kasus

Buatlah sebuah function dengan nama `myListFunction()` yang memiliki parameter dalam bentuk list. Misalkan diberikan suatu list yaitu:

```
buah = ['apel', 'durian', 'jeruk', 'klengkeng', 'mangga']
```

maka jika list buah tersebut dimasukkan ke dalam function, atau `myListFunction(buah)`, function harus menampilkan output sbb:

```
'apel, durian, jeruk, klengkeng, dan mangga'
```

### Jawab:

Salah satu contoh program sebagai solusi dari permasalahan di atas adalah sebagai berikut.

```
def myListFunction(myList):
    for i in range(len(myList)):
        if i == len(myList)-1:
            print('dan ' + myList[-1])
        else:
            print(myList[i] + ', ', end='')

myList = ['apel', 'jeruk', 'mangga']
myListFunction(myList)
```

Dalam program di atas, ide penyelesaiannya adalah dengan melakukan looping sebanyak jumlah nilai pada listnya (digunakan `len(myList)` untuk membaca jumlah nilai dalam list `myList`). Selanjutnya di setiap looping, nilai `i` nya dicek apakah nilainya sama dengan `len(myList)-1` atau tidak. Jika ya maka ditampilkan 'dan' dan nilai pada indeks terakhirnya (di sini digunakan indeks `-1`). Hal ini dilakukan karena khusus untuk indeks terakhir (`len()-1`), ada perbedaan tampilan yaitu adanya tambahan kata 'dan' di depannya. Sedangkan untuk `i` yang lainnya, maka cukup ditambahkan tanda koma di belakangnya.

### Studi Kasus

Buatlah sebuah function dengan nama `printGrid()` yang memiliki parameter dalam bentuk list. Misalkan diberikan suatu list yaitu:

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Maka apabila dipanggil perintah `printGrid(grid)` akan muncul tampilan sebagai berikut:

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO.
...OOO...
....O....
```

#### Jawab:

Apabila list dua dimensi yang diberikan pada kasus tersebut kita lihat sebagai matriks, maka permasalahan yang diberikan ini merupakan permasalahan terkait dengan transpose matrik. Transpose matrik didapat dengan menukar posisi indeks yg terkait dengan nomor baris dengan kolom, begitu juga sebaliknya. Sehingga misalkan indeks ke  $[i, j]$  di mana  $i$  adalah indeks terkait nomor baris dan  $j$  adalah nomor kolom dari sebuah nilai pada matriks awalnya, maka pada transpose matriknya nilai tersebut menjadi indeks ke  $[j, i]$ , di mana  $j$  menjadi indeks terkait nomor baris dan  $i$  terkait dengan nomor kolom.

```
def printGrid(theList):
    baris = len(theList)           # jumlah baris
    kolom = len(theList[0])        # jumlah kolom
    for i in range(kolom):
        for j in range(baris):
            print(theList[j][i], end='')
        print()

grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]

printGrid(grid)
```

Perhatikan program di atas, untuk mengetahui jumlah kolom dari grid dengan cara membaca banyaknya nilai pada indeks pertama dari list nya (`len(theList[0])`). Meskipun demikian, sebenarnya tidak harus pada indeks pertama, akan tetapi bisa juga dari indeks yang lainnya.

## Soal Latihan

### Dictionaries

Di dalam Python, tipe data dictionaries mirip seperti list. Namun perbedaannya hanyalah pada index nya saja. Seperti yang telah kita pahami bahwa index dari list adalah berupa nomor atau bilangan bulat dan indexnya terurut. Sedangkan dictionaries, index nya tidak hanya berupa bilangan bulat saja namun bisa dengan yang lainnya, tidak harus secara urut.

Sebuah variabel bertipe dictionaries ditandai dengan tanda kurung `{ }` dan di dalam kurung terdapat satu atau lebih pasangan key dan value

```
{key1 : value1, key2 : value2, ... }
```

Adapun cara untuk membaca nilai di dalam dictionaries sama seperti list, yaitu cukup disebutkan nama indeksnya.

Perhatikan contoh pendefinisian suatu variabel bertipe data dictionaries beserta cara membaca nilai pada indeks tertentu:

```
>>> tinggi = {'Rosihan Ari' : 170, 'Siti Nurhaliza' : 160, 'Budiman' : 165}
>>> tinggi['Rosihan Ari']
170
```

Pada contoh di atas, tipe data dictionaries digunakan untuk menyatakan data tinggi badan pada nama seseorang.

```
>>> mobilku = {'merk' : 'Fortuner', 'warna' : 'hitam', 'transmisi' : 'matic', 'tahun' : 2015}
>>> mobilku['transmisi']
'matic'
```

Sedangkan pada contoh di atas, dictionaries digunakan untuk menyatakan properti dan nilai dari suatu obyek. Contoh yang diambil di atas adalah properti dari suatu obyek berupa mobil, yaitu merek, warna, jenis transmisi, dan tahun pembuatannya.



Selanjutnya akan diberikan sebuah contoh program Python yang memanfaatkan dictionaries.

```
# data harga buah
hargaBuah = {'jeruk' : 15000, 'apel' : 25000, 'melon' : 18500,
'anggur' : 42000}

while True:
    print('Masukkan nama buah : ', end='')
    nama = input()
    # jika nama buah blank maka exit
    if nama == '':
        break

    # jika nama buah ada di data maka tampilkan harga
    if nama in hargaBuah:
        print('Harganya          : Rp ' + str(hargaBuah[nama]))
    else:
        # jika tidak terdaftar maka masukkan ke data
        print('Nama buah ' + nama + ' belum ada di daftar harga.')
        print('Berapa harganya    : ', end='')
        harga = int(input())
        # memasukkan harga dan nama buah ke data
        hargaBuah[nama] = harga
        print('Database harga telah diupdate')
```

Program di atas merupakan implementasi dari dictionaries untuk membuat aplikasi database sederhana, pada kasus data harga buah. Di sini dibuat data berisi nama buah dan harganya menggunakan dictionaries. Selanjutnya user diminta memasukkan nama buahnya. Apabila nama buah yang dimasukkan user ini ada di data, maka akan ditampilkan harganya. Namun apabila tidak terdaftar di data maka buah dan harga akan dimasukkan ke data (mengupdate data).

### Perintah keys ( )

Perintah keys() di dalam dictionary biasanya digunakan di dalam looping yang akan membaca seluruh key dalam dictionary tersebut. Contoh:

```
mhs1 = {'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' :
'01/09/1979'}

for x in mhs1.keys():
    print(x)
```

Output dari contoh di atas adalah:

```
nim
nama
tgllhr
```

Berdasarkan contoh di atas, setiap key yang dibaca selanjutnya disimpan ke dalam variabel x.

### Perintah `values()`

Sedangkan untuk `values()` digunakan untuk membaca seluruh value dalam dictionary. Adapun contohnya adalah:

```
mhs1 = {'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' : '01/09/1979'}
for x in mhs1.values():
    print(x)
```

Outputnya adalah:

```
01/09/1979
M0197065
Rosihan Ari
```

Perlu diketahui bahwa output hasil `values()` memiliki urutan yang teracak. Sehingga di dalam kasus tertentu, apabila diinginkan mendapatkan values yang terurut maka terlebih dahulu dinyatakan dalam bentuk list kemudian dilakukan proses sorting.

```
mhs1 = {'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' : '01/09/1979'}

# mengkonversi values dictionary ke list
data = list(mhs1.values())
# lakukan sorting
data.sort()

for x in data:
    print(x)
```

### Perintah `items()`

Perintah `items()` di dalam dictionary digunakan untuk membaca semua pasangan key dan value.

Contohnya:

```
mhs1 = {'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' : '01/09/1979'}

for x in mhs1.items():
    print(x)
```

dan outputnya adalah

```
( 'nama', 'Rosihan Ari')
( 'tgllhr', '01/09/1979')
( 'nim', 'M0197065')
```

Jika dalam contoh di atas nilai x berisi pasangan key dan value, maka terdapat cara lain untuk membaca key dan value secara sendiri-sendiri menggunakan dua variabel. Contoh:

```
mhs1 = { 'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' :
'01/09/1979' }

for x, y in mhs1.items():
    print('Key : ' + x + ', Value : ' + y)
```

Outputnya adalah sebagai berikut:

```
Key : nama, Value : Rosihan Ari
Key : nim, Value : M0197065
Key : tgllhr, Value : 01/09/1979
```

### Mengecek Key atau Value dalam Dictionary

Seperti halnya list, di dalam dictionary juga bisa digunakan perintah `in` dan `not in` untuk mengecek keberadaan key atau value. Contoh:

```
>>> mhs1 = { 'nama' : 'Rosihan Ari', 'nim' : 'M0197065', 'tgllhr' :
'01/09/1979' }
>>> 'nama' in mhs1.keys()
True
>>> 'alamat' in mhs1.keys()
False
>>> 'M0197065' in mhs1.values()
True
>>> 'Budi' not in mhs1.values()
True
```

### Perintah `get()`

Misalkan diberikan contoh program berikut ini:

```
dataBuah = { 'apel' : 3, 'jeruk' : 5, 'duku' : 10 }

buah = 'apel';

if buah in dataBuah.keys():
    print('Banyaknya ' + buah + ' : ' + str(dataBuah[buah]))
else:
    print('Banyaknya ' + buah + ' : ' + str(0))
```

Pada program di atas, misalkan ada dictionary berupa data buah yang di dalamnya terdapat nama buah dan jumlahnya. Selanjutnya, di baris berikutnya akan ditampilkan jumlah buah berdasarkan nama buahnya. Jika buahnya 'apel' maka akan muncul output:

```
Banyaknya apel : 3
```

Sedangkan apabila nama buahnya tidak terdapat di dictionary, maka akan menghasilkan jumlah 0. Misalnya:

```
dataBuah = {'apel' : 3, 'jeruk' : 5, 'duku' : 10}
buah = 'semangka';
if buah in dataBuah.keys():
    print('Banyaknya ' + buah + ' : ' + str(dataBuah[buah]))
else:
    print('Banyaknya ' + buah + ' : ' + str(0))
```

Outputnya:

```
Banyaknya semangka : 0
```

Dalam hal ini, untuk kasus di atas digunakan statement IF. Meskipun demikian terdapat perintah yang lebih sederhana untuk menyederhanakan program di atas, khususnya menggantikan statement IF, yaitu menggunakan perintah get(). Contohnya adalah:

```
dataBuah = {'apel' : 3, 'jeruk' : 5, 'duku' : 10}
buah = 'apel';
print('Banyaknya ' + buah + ' : ' + str(dataBuah.get(buah, 0)))
```

Output dari program di atas adalah:

```
Banyaknya apel : 3
```

Sedangkan jika nama buahnya tidak terdapat dalam dictionary akan misalnya

```
dataBuah = {'apel' : 3, 'jeruk' : 5, 'duku' : 10}
buah = 'mangga'
print('Banyaknya ' + buah + ' : ' + str(dataBuah.get(buah, 0)))
```

akan menghasilkan:

```
Banyaknya mangga : 0
```

Berdasarkan contoh di atas, maka perintah `get(x, y)` digunakan untuk membaca value dalam dictionary yang memiliki key = x. Apabila key tersebut tidak ada, maka value yang dibaca (default) adalah y.

## Manipulasi String

String di dalam Python dapat ditulis dengan diapit single quotes (tanda petik tunggal) maupun double quote (tanda petik ganda).

Kelebihan dari penggunaan tanda petik ganda adalah kita bisa menyisipkan tanda petik tunggal di dalam string tersebut.

```
>>> nama = "Siti 'Aisah"
>>> print(nama)
Siti 'Aisah
```

Apabila kita menggunakan tanda petik tunggal dan kita ingin menyisipkan tanda petik tunggal juga di dalam stringnya, maka caranya dengan menambahkan backslash di depan tanda petiknya.

```
>>> nama = 'Siti \'Aisah'
>>> print(nama)
Siti 'Aisah
```

## Escape Characters

Escape characters adalah karakter-karakter yang memiliki makna khusus. Berikut ini di antaranya:

Karakter	Makna
<code>\n</code>	Baris baru (new line)
<code>\t</code>	Tab
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\\</code>	Backslash

Contoh penggunaan:

```
>>> myString = "Hello\nWorld"
>>> print(myString)
Hello
World
```

```
>>> myString = "Hello\tWorld"
>>> print(myString)
Hello World
```

## Operator in dan not in dalam String

Seperti halnya list, tuple, dan dictionary, untuk string juga bisa digunakan operator in dan not in. Operator ini digunakan untuk mengecek apakah suatu substring terdapat dalam string tersebut.

Contoh:

```
>>> myString = 'Hello World, I\'m Fine'
>>> 'World' in myString
True
>>> 'world' not in myString
True
>>> 'hello' in myString
False
```

## Perintah upper() dan lower()

Perintah upper() digunakan untuk mengubah string menjadi huruf kapital semuanya. Sedangkan lower() untuk mengubah string ke huruf kecil. Contohnya:

```
>>> myString = 'Hello World, I\'m Fine'
>>> myString.upper()
"HELLO WORLD, I'M FINE"
>>> myString.lower()
"hello world, i'm fine"
```

## Perintah join()

Selain menggunakan operator string concatenation (+), kita juga bisa melakukan penggabungan string dengan perintah join(). Kelebihan dari join() adalah kita bisa menentukan karakter yang menjadi penghubung antar string yang digabung. Contoh:

```
>>> gabung = ', '.join(['mangga', 'apel', 'rambutan'])
>>> gabung
'mangga, apel, rambutan'

>>> gabung = ' dan '.join(['mangga', 'apel', 'rambutan'])
>>> gabung
'mangga dan apel dan rambutan'

>>> gabung = ' '.join(['mangga', 'apel', 'rambutan'])
>>> gabung
'mangga apel rambutan'
```

## Perintah split()

Kebalikan dari join(), perintah split() digunakan untuk memecah string menjadi beberapa substring berdasarkan karakter tertentu. Contohnya:

```
>>> buah = 'mangga dan apel dan durian'.split('dan')
>>> buah
['mangga ', ' apel ', ' durian']

>>> buah = 'mangga apel durian'.split()
>>> buah
['mangga', 'apel', 'durian']
```

Dari contoh di atas dapat kita lihat bahwa secara default perintah split() memecah string berdasarkan karakter spasi (whitespace). Selain itu kita bisa lihat pula bahwa hasil split() adalah sebuah list berisi substring.

## Pengaturan Justifikasi String

Di dalam Python, kita dapat dengan mudah untuk mengatur justifikasi posisi string ketika ditampilkan dengan perintah print(). Terdapat tiga perintah cara pengaturan justifikasi string yaitu: rjust(), ljust(), center().

### Perintah rjust()

Perintah rjust() digunakan untuk mengatur string dalam posisi rata kanan relatif terhadap space yang ditentukan. Contoh:

```
>>> buah = 'mangga'
>>> print(buah.rjust(20))
                mangga

>>> print(buah.rjust(20, '.'))
.....mangga
```

Berdasarkan contoh di atas, terdapat 20 space untuk mencetak string 'mangga'. Pada contoh yang pertama, posisi string 'mangga' berada di paling kanan dari space yang disediakan dan terdapat spasi kosong di depan string tersebut. Sedangkan pada contoh ke dua, di depan string terdapat karakter '.'.

### Perintah ljust()

Kebalikan dari rjust(), perintah ljust() digunakan untuk perataan kiri dari string relatif terhadap space yang diberikan. Contohnya:

```
>>> buah = 'mangga'
>>> print(buah.ljust(20))
mangga
>>> print(buah.ljust(20, '*'))
mangga*****
```

### Perintah center()

Sedangkan perintah center() digunakan untuk memposisikan string di tengah-tengah space yang diberikan. Contoh:

```
>>> buah = 'mangga'
>>> buah.center(20)
'      mangga      '
>>> buah.center(20, '=')
'=====mangga====='
```