Modul Praktikum

Pemrograman Interpreter

Dosen Pengampu :
Dr. Ir. Kurnia Muludi, M.S.Sc

python

Disusun Oleh:

Aulia Ahmad Nabil

Laboratorium Komputasi Dasar
Jurusan Ilmu Komputer
FMIPA Universitas Lampung

Pengenalan Python

Python adalah bahasa pemrograman multifungsi yang dibuat oleh Guido van Rossum dan dirilis pada tahun 1991. GvR, begitu ia biasa disebut di komunitas Python, menciptakan Python untuk menjadi interpreter yang memiliki kemampuan penanganan kesalahan (exception handling) dan mengutamakan sintaksis yang mudah dibaca serta dimengerti (readability). Didesain untuk memudahkan dalam prototyping, Python menjadi bahasa yang sangat mudah dipahami dan fleksibel.

Python juga memilih untuk menggunakan indentasi untuk mengelompokkan blok kode, berbeda dengan beberapa bahasa lain yang menggunakan simbol tertentu, misalnya kurung kurawal, atau sintaksis begin-end. Sehingga secara visual pun, blok kode Python didesain untuk mudah dipahami. Salah satu yang paling dikenal adalah, penggunaan titik koma atau semicolon (;) tidak wajib di Python dan penggunaan semicolon cenderung dianggap bukan cara khas Python (non-pythonic way), meskipun ia tetap dapat digunakan, misalnya untuk memisahkan dua statement dalam baris yang sama.

```
.   print("Hello World"); print("Welcome to Python")
```

Python juga memilih untuk mengadopsi *dynamic typing* secara opsional, yakni variabel yang dibuat tidak akan diketahui tipenya hingga ia dipanggil pertama kali atau dieksekusi, tidak perlu deklarasi variabel (meskipun dimungkinkan), dan memungkinkan tipe data berubah dalam proses eksekusi program. Sejak Python versi 3.6, sudah tersedia pilihan untuk *static typing*.

Mengapa Python?

Efektivitas Python cukup terbukti dengan banyaknya jumlah pengguna Bahasa Pemrograman ini. Berbagai survei memasukkan Python dalam top-3 sebagai bahasa dengan penggunaan terbanyak, bersaing dengan Java dan PHP. Python dapat digunakan dalam mengakomodasi berbagai gaya pemrograman, termasuk structured, prosedural, berorientasi-objek, maupun fungsional. Python juga dapat berjalan pada berbagai sistem operasi yang tersedia. Beberapa pemanfaatan bahasa Python di antaranya:

1. Web development (server-side),
2. Software development,
3. Mathematics & data science,
4. Machine learning,
5. System scripting.
6. Internet of Things (IoT) development.

Persiapan

Pada Linux dan Mac, umumnya Python sudah terinstal secara otomatis. Untuk memastikan python sudah terinstal atau memeriksa versi python yang terinstal, silakan panggil perintah berikut di konsol atau command prompt:

```
python --version
```

atau

```
python3 --version
```

Jika telah terinstal, maka akan tampil versi Python yang terinstal sebagai berikut (Klik tab pada tabel untuk melihat tampilan di masing-masing sistem operasi).

Lakukan instalasi, pastikan mencentang Add Python 3.7 to PATH untuk menambahkan Python dalam Environment Variables.

IDE

Integrated Development Environment (IDE) lebih dari sekedar text editor untuk membuat kode, di dalamnya tergabung juga berbagai fasilitas development misalnya Code Versioning, Interpreter, Visualization, dan lain sebagainya.

Mode pada Python

Pada Python dikenal beberapa mode operasi: Interactive, Script (scripting), dan Notebook.

Interactive

Berbeda dengan bahasa pemrograman lainnya, bahasa Python yang berbasis interpreter memungkinkan kita untuk menjalankan perintah secara interaktif. Mode ini dapat diakses di bagian bawah PyCharm atau dengan memanggil perintah python di command prompt/terminal.

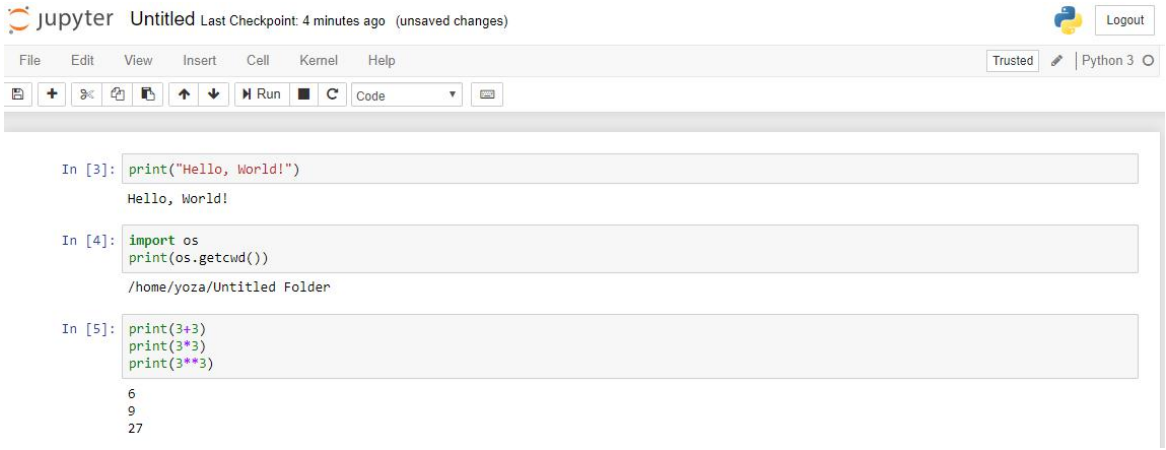
```
yoza@beaver:~$ python
Python 3.7.3 (default, Mar 26 2019, 00:55:50)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+3
6
>>> 3*3
9
>>> 3**3
27
>>>
```

Script

Mode yang lain dan sering dipergunakan pada python adalah script (scripting). Pada mode ini kita menggunakan sebuah berkas teks (umumnya berekstensi .py) dan kemudian akan dieksekusi oleh compiler/interpreter. Contoh yang sama untuk ketiga hal yang kita coba pada mode interactive adalah sebagai berikut:

```
1  print(3+3)
2  print(3*3)
3  print(3**3)
```

Notebook



Style Guide pada Python Code

Berikut adalah beberapa style guide menulis kode Python dengan baik dan benar. Panduan gaya penulisan kode ini mengacu pada PEP-008. Beberapa proyek mungkin memiliki style guide tersendiri. Sejumlah contoh kode yang ditulis di halaman ini berupa pseudocode, bertujuan hanya untuk memberikan gambaran tentang panduan gaya penulisan kode saja.

Guido, pembuat bahasa Python, merasakan bahwa kode lebih sering dibaca dibandingkan ditulis. Oleh sebab itu, panduan ini lebih ditekankan untuk kemudahan membaca kode dan membuatnya konsisten pada (hampir) setiap proyek Python yang ada.

Namun demikian pada kasus-kasus tertentu, keputusan adanya modifikasi tetap pada penulis kodenya. Mungkin sebuah kode dapat terbaca lebih jelas walaupun tidak mengikuti satu atau lebih panduan dalam modul ini.

Indentasi

Gunakan 4 spasi pada setiap tingkatan indentasi.

Python menggunakan indentasi untuk menulis kode bertingkat. Bahasa lain mungkin menggunakan statement tertentu (Begin, end - pascal), perbedaan baris atau kurung kurawal. Statement yang memiliki indentasi yang sama dan diletakkan secara berurutan dikenali sebagai blok statement oleh Python dan akan dijalankan secara berurutan.

```
.   Statement tingkat 1:
.       Statement tingkat 2()
.       Statement tingkat 2 yang kedua()
```

Tipe Data

Dasar dari mempelajari Bahasa Pemrograman yang baru adalah pemahaman terhadap tipe data. Di sini Anda akan diajarkan tentang tipe data bawaan yang ada di Python 3 beserta contoh penggunaannya.

Numbers

Tipe numerik pada Python dibagi menjadi 3: int, float, complex.

```
a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Integer tidak dibatasi oleh angka atau panjang tertentu, namun dibatasi oleh memori yang tersedia. Sehingga Anda tidak perlu menggunakan variabel yang menampung big number misalnya long long (C/C++), biginteger, atau sejenisnya.

Strings

String adalah urutan dari karakter unicode yang dideklarasikan dengan petik tunggal atau ganda. String >1baris dapat ditandai dengan tiga petik tunggal atau ganda ''' atau """.

```
s = "This is a string"

s = '''this is a multiline
next new line (1)
another new line (2)'''
```

Bool/Boolean

Tipe data bool atau Boolean merupakan turunan dari bilangan bulat (integer atau int) yang hanya punya dua nilai konstanta: True dan False.

List

List adalah jenis kumpulan data terurut (ordered sequence), dan merupakan salah satu variabel yang sering digunakan pada Python. Serupa, namun tak sama dengan array pada bahasa pemrograman lainnya. Bedanya, elemen List pada Python tidak harus memiliki tipe data yang sama. Mendeklarasikan List cukup mudah dengan kurung siku dan elemen yang dipisahkan dengan koma. Setiap data didalamnya dapat diakses dengan indeks yang dimulai dari 0.

```
a = [1, 2.2, 'python']
```

```
x = [5,10,15,20,25,30,35,40]
print(x[5])
print(x[-1])
print(x[3:5])
print(x[:5])
print(x[-3:])
print(x[1:7:2])
```

Elemen pada list dapat diubah atau ditambahkan. Misalnya untuk melakukan perubahan kemudian penambahan:

```
x = [1,2,3]
x[2]=4
```

```
x.append(5)
```

Untuk menghapus, gunakan fungsi del.

```
spam = ['cat', 'bat', 'rat', 'elephant']
del spam[2]
```

Tuple

Tuple adalah jenis dari list yang tidak dapat diubah elemennya. Umumnya tuple digunakan untuk data yang bersifat sekali tulis, dan dapat dieksekusi lebih cepat. Tuple didefinisikan dengan kurung dan elemen yang dipisahkan dengan koma.

```
t = (5,'program', 1+3j)
```

Set

Set adalah kumpulan item bersifat unik dan tanpa urutan (unordered collection). Didefinisikan dengan kurawal dan elemennya dipisahkan dengan koma. Pada Set kita dapat melakukan union dan intersection, sekaligus otomatis melakukan penghapusan data duplikat.

```
a = {1,2,2,3,3,3}
```

Karena set bersifat unordered, maka kita tidak bisa mengambil sebagian data / elemen datanya menggunakan proses slicing.

```
a = {1,2,3}
a[1]
```

Dictionary

Dictionary pada Python adalah kumpulan pasangan kunci-nilai (pair of key-value) yang bersifat tidak berurutan. Dictionary dapat digunakan untuk menyimpan data kecil hingga besar. Untuk mengakses datanya, kita harus mengetahui kuncinya (key). Pada Python, dictionary didefinisikan dengan kurawal dan tambahan definisi berikut:

1. Setiap elemen pair key-value dipisahkan dengan koma (,).
2. Key dan Value dipisahkan dengan titik dua (:).
3. Key dan Value dapat berupa tipe variabel/obyek apapun.

```
d = {1:'value','key':2}
```

```
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);
```

Dictionary bukan termasuk dalam implementasi urutan (sequences), sehingga tidak bisa dipanggil dengan urutan indeks. Misalnya dalam contoh berikut dicoba dengan indeks 2, tetapi menghasilkan error (KeyError) karena tidak ada kunci (key) 2:

```
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

Input/Output

Di bagian ini Anda akan mempelajari tentang mekanisme Input/Output, misalnya meminta masukan dari pengguna, menyimpan nilai pada variabel dan mencetak nilai ke layar.

Setelah sebelumnya mempelajari tipe data, selanjutnya Anda akan belajar tentang variabel. Variabel adalah sebuah tempat (di memori komputer) untuk menyimpan nilai dengan tipe data tertentu.

Untuk memberikan nilai pada sebuah variabel, kita menggunakan operator "=", antara nama variabel dengan nilai yang ingin disimpan.

Misalnya: x = 1.

Artinya kita akan menyimpan nilai 1 (tipe int) ke variabel x.

Output

Print

Seperti dicontohkan dalam beberapa sample code sebelumnya, fungsi print() adalah cara output langsung ke konsol/layar.

```
print("Hello, World!")
```

Memasukkan nilai variabel pada string

Untuk memasukkan nilai variabel pada string, Python memiliki dua cara. Cara yang pertama adalah langsung menggabungkan variabel pada statement print().

```
a = 5
print('The value of a is', a)
```

Output:

The value of a is 5

Untuk menampilkan text (string), bisa menggunakan mekanisme string format. Misalnya yang pertama:

```
print('hello {}'.format('kakak'))
```

Cara yang kedua mirip dengan sintaks C/C++, yakni menggunakan operator "%" yang ditambahkan dengan "argument specifiers", misalnya "%s" and "%d". Contohnya saat kita ingin menambahkan nama kita pada string hello:

```
name = "John"
print("Hello, %s!" % name)
```

Output:

Hello, John!

Contoh menambahkan string dan integer:

```
name = "John"
age = 15
print("%s is %d years old." % (name, age))
```

Output:

John is 15 years old.

Beberapa argument specifier yang umum digunakan:

```
%s - String
%d - Integers
%f - Bilangan Desimal
%.<digit>f - Bilangan desimal dengan sejumlah digit angka dibelakang koma.
%x/%X - Bilangan bulat dalam representasi Hexa (huruf kecil/huruf besar)
```

Input

input()

Untuk memungkinkan user memberikan input pada program Anda, gunakan fungsi input(), dengan argumen dalam kurung () adalah teks yang ingin ditampilkan (prompt) dan variabel sebelum tanda sama dengan (=) adalah penampung hasil dari input pengguna:

```
num = input('Enter a number: ')
```

Secara default, input dari user adalah string (walaupun pada contoh di atas, 10 sebenarnya dimaksudkan sebagai integer) yang ditandai dengan petik. Untuk itu diperlukan fungsi konversi yang akan dibahas pada modul-modul selanjutnya, misalnya int(), float() atau eval().

Strings

String adalah urutan dari karakter unicode yang dideklarasikan dengan petik tunggal atau ganda. String >1baris dapat ditandai dengan tiga petik tunggal atau ganda ''' atau """.

```
s = "This is a string"

s = '''this is a multiline
next new line (1)
another new line (2)'''
```

Seperti list dan tuple, slicing operator [ ] dapat digunakan pada string. Sebuah string utuh bersifat mutable (bisa diubah), namun elemennya bersifat immutable (tidak bisa diubah).

```
s = "Hello World!"
print(s[4])
print(s[6:11])
s[5]="d"
```

Output:

```
'o'
'World'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
s = "Hello World!"
print(s)
s= "Try Python!"
print(s)
```

Output:

```
'Hello World!'
'Try Python!'
```

Transformasi karakter dan string

Metode upper() dan lower() dari String (dan karakter)

Method upper() dan lower() adalah metode konversi untuk membuat seluruh karakter dalam string menjadi kapital (upper) atau huruf kecil (lower). Jika terdapat karakter non huruf yang tidak memiliki opsi kapital, maka karakter tersebut tidak diubah.
Contoh:

```
p = 'Hello world!'
p = p.upper()
p
```

Output:

```
'HELLO WORLD!'
```

```
p = p.lower()
p
```

Output:

```
'hello world!'
```

Berbeda dengan fungsi sort() yang mengubah variabel secara langsung, lower() dan upper() akan mengembalikan string baru. Sehingga Anda perlu menampungnya dalam variabel.
upper() dan lower() umumnya digunakan untuk memiliki perbandingan yang bersifat case-insensitive. 'Dicoding', 'DIcoding', atau 'DICODING' tidak sama satu dengan lainnya, namun Anda bisa memastikan karakternya sama jika Anda menggunakan upper() atau lower() saat membandingkan:
Cobalah menjalankan kode berikut:

```
feeling = input('How are you?')
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

isupper() dan islower()

Sementara isupper() dan islower() akan mengembalikan nilai boolean jika string yang dimaksud memiliki satu karakter dan seluruhnya kapital atau seluruhnya huruf kecil. Jika syarat tidak terpenuhi, maka fungsi/method akan mengembalikan nilai false. Contoh:

```
p = 'Hello world!'
p.islower()
```

Output:

```
False
```

```
p.isupper()
```

Output:

```
False
```

```
'HELLO'.isupper()
```

Output:

```
True
```

```
'abc12345'.islower()
```

Output:

```
True
```

```
'12345'.islower()
```

Output:

```
False
```

```
'12345'.isupper()
```

Output:

```
False
```

Karena fungsi upper() dan lower() mengembalikan string, maka Anda dapat melakukan operasi pada hasil operasinya (chain of method), contohnya:

```
. print('Hello'.upper())
. print('Hello'.upper().lower())
. print('Hello'.upper().lower().upper())
. print('HELLO'.lower())
. print('HELLO'.lower().islower())
```

Output:

```
'HELLO'
'hello'
'HELLO'
'hello'
True
```

Metode isX dari String untuk Pengecekan

Selain islower() and isupper(), terdapat beberapa metode lain yang dapat digunakan untuk memeriksa isi dari string. Semua method berikut mengembalikan nilai boolean:

- isalpha() mengembalikan True jika string berisi hanya huruf dan tidak kosong.
- isalnum() mengembalikan True jika string berisi hanya huruf atau angka, dan tidak kosong.
- isdecimal() mengembalikan True jika string berisi hanya angka/numerik dan tidak kosong.
- isspace() mengembalikan True jika string berisi hanya spasi, tab, newline, atau whitespaces lainnya dan tidak kosong.
- istitle() mengembalikan True jika string berisi kata yang diawali huruf kapital dan dilanjutkan dengan huruf kecil seterusnya.

Contoh:

```
. 'hello'.isalpha()
```

Output:

```
True
```

```
. 'hello123'.isalpha()
```

Output:

```
False
```

```
. 'hello123'.isalnum()
```

Output:

```
True
```

```
. 'hello'.isalnum()
```

Output:

```
True
```

```
. '123'.isdecimal()
```

Output:

```
True
```

```
. '    '.isspace()
```

Output:

```
True
```

```
. 'This Is Title Case'.istitle()
```

Output:

```
True
```

```
. 'This Is Title Case 123'.istitle()
```

Output:

```
True
```

```
. 'This Is not Title Case'.istitle()
```

Output:

```
False
```

```
. 'This Is NOT Title Case Either'.istitle()
```

Output:

```
False
```

Metode isX akan membantu misalnya dalam validasi input user. Anda bisa mencobanya dengan contoh berikut. Bukalah file baru pada notepad atau IDE, dan simpan sebagai validateInput.py. Pastikan Anda menggunakan Python 3 untuk mencobanya (akan ada error saat menggunakan Python 2.7).

```
. while True:
.     print('Enter your age:')
```

```
.       age = input()
.       if age.isdecimal():
.           break
.       print('Please enter a number for your age.')
.   while True:
.       print('Select a new password (letters and numbers
    only):')
.       password = input()
.       if password.isalnum():
.           break
.       print('Passwords can only have letters and number
    s.')
```

Pada perulangan while yang pertama, program akan meminta usia pengguna. Jika valid, maka program akan berlanjut ke perulangan while kedua yang meminta password dengan spesifikasi alphanumeric (angka dan huruf). Ketika dijalankan akan seperti berikut:
Output:

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
Secr3t
```

Dengan cukup memanggil isdecimal() dan isalnum(), Anda dapat menyingkat waktu untuk melakukan validasi input user.

Metode startswith() dan endswith() dari String

Fungsi startswith() dan endswith() akan mengembalikan nilai True berdasarkan nilai awalan atau akhiran string. Contohnya sebagai berikut:

```
. 'Hello world!'.startswith('Hello')
```

Output:

```
True
```

```
. 'Hello world!'.endswith('world!')
```

Output:

```
True
```

```
. 'abc123'.startswith('abcdef')
```

Output:

```
False
```

```
. 'abc123'.endswith('12')
```

Output:

```
False
```

```
. 'Hello world!'.startswith('Hello world!')
```

Output:

```
True
```

```
. 'Hello world!'.endswith('Hello world!')
```

Output:

```
True
```

Fungsi/method ini akan sangat efisien jika Anda hanya perlu membandingkan awal atau akhir string, tanpa perlu melakukan splitting dan menggunakan statement perbandingan sama dengan (==).

Metode join() dan split() dari String

Fungsi join() berguna saat Anda memiliki sejumlah string yang perlu digabungkan. Contohnya sebagai berikut:

```
', '.join(['cats', 'rats', 'bats'])
```

Output:

```
'cats, rats, bats'
```

```
' '.join(['My', 'name', 'is', 'Simon'])
```

Output:

```
'My name is Simon'
```

```
'ABC'.join(['My', 'name', 'is', 'Simon'])
```

Output:

```
'MyABCnameABCisABCSimon'
```

Perhatikan bahwa string yang dioperasi dengan fungsi join() akan ditambahkan/disisipkan di antara setiap parameter/argument. Misalnya koma pada 'cats, rats, bats'. Sebaliknya metode split() memisahkan substring berdasarkan delimiter tertentu (defaultnya adalah whitespace - spasi, tab, atau newline):

```
'My name is Simon'.split()
```

Output:

```
['My', 'name', 'is', 'Simon']
```

Anda dapat mengubah parameter split (delimiter) seperti berikut:

```
'MyABCnameABCisABCSimon'.split('ABC')
```

Output:

```
['My', 'name', 'is', 'Simon']
```

```
'My name is Simon'.split('m')
```

Output:

```
['My na', 'e is Si', 'on']
```

Salah satu penggunaan paling sering dari split() adalah memisahkan setiap baris pada string multiline:

```
. a = '''Dear Alice,
. How have you been? I am fine.
. There is a container in the fridge
. that is labeled "Milk Experiment".
. Please do not drink it.
. Sincerely,
. Bob'''
. a.split('\n')
```

Output:

```
['Dear Alice,',
'How have you been? I am fine.',
'There is a container in the fridge',
'that is labeled "Milk Experiment".',
'Please do not drink it.',
'Sincerely,',
'Bob']
```

Teks rata kanan/kiri/tengah dengan rjust(), ljust(), dan center()

Anda dapat merapikan pencetakan teks di layar dengan rjust(), ljust() atau center(). rjust() dan ljust() akan menambahkan spasi pada string untuk membuatnya sesuai (misalnya rata kiri atau rata kanan). Argumennya berupa integer yang merupakan panjang teks secara keseluruhan (bukan jumlah spasi yang ditambahkan):

```
. 'Hello'.rjust(10)
```

Output:

```
'     Hello'
```

```
. 'Hello'.rjust(20)
```

Output:

```
'               Hello'
```

```
. 'Hello World'.rjust(20)
```

Output:

```
'          Hello World'
```

```
>>> 'Hello'.ljust(10)
```

Output:

```
'Hello     '
```

Contohnya: 'Hello'.rjust(10) dapat diartikan sebagai kita ingin menuliskan Hello dalam mode rata kanan dengan total panjang string 10. Karena panjang 'Hello' adalah 5 karakter, maka 5 spasi akan ditambahkan di sebelah kiri. Selain spasi, Anda juga bisa menambahkan karakter lain dengan mengisikan parameter kedua pada fungsi rjust() atau ljust():

```
>>> 'Hello'.rjust(20, '*')
```

Output:

```
'***************Hello'
```

```
>>> 'Hello'.ljust(20, '-')
```

Output:

```
'Hello---------------'
```

Metode center() seperti namanya akan membuat teks Anda rata tengah:

```
>>> 'Hello'.center(20)
```

Output:

```
'       Hello        '
```

```
>>> 'Hello'.center(20, '=')
```

Output:

```
'=======Hello========'
```

Jika Anda memprogram aplikasi berbasis konsol (CLI), maka fungsi-fungsi di atas akan sangat berguna saat membuat tabulasi/tabel.

Hapus Whitespace dengan strip(), rstrip(), dan lstrip()

Saat Anda menerima string sebagai parameter, seringkali masih terdapat karakter whitespace (spasi, tab, dan newline) di bagian awal dan atau akhir string yang dimaksud. Metode strip() akan menghapus whitespace pada bagian awal atau akhir string. lstrip() dan rstrip() akan menghapus sesuai dengan namanya, awal saja atau akhir saja:

```
>>> spam = '    Hello World     '
```

```
spam.strip()
```

Output:

```
'Hello World'
```

```
spam.lstrip()
```

Output:

```
'Hello World     '
```

```
spam.rstrip()
```

Output:

```
'     Hello World'
```

Anda juga bisa menentukan mana karakter atau bagian yang ingin dihilangkan, misalnya:

```
spam = 'SpamSpamBaconSpamEggsSpamSpam'
spam.strip('ampS')
```

Output:

```
'BaconSpamEggs'
```

Saat mengirimkan 'ampS' sebagai argumen strip, maka Python akan menghapus setiap huruf a, m, p dan S kapital dari string yang dioperasikan. Urutan untuk parameter ini tidak berpengaruh, sehingga strip('ampS') akan berlaku sama dengan ('mapS') atau strip('Spam').

Mengganti string/substring dengan replace()

replace() adalah satu fungsi python yang mengembalikan string baru dalam kondisi substring telah tergantikan dengan parameter yang dimasukkan:

```
string = "Ayo belajar Coding di Dicoding"
print(string.replace("Coding", "Pemrograman"))
```

Output:

```
Ayo belajar Pemrograman di Dicoding
```

Perhatikan bahwa "coding" di frase "Dicoding" tidak berubah, maka fungsi replace bersifat Case Sensitive.

Opsional, Parameter ketiga pada replace dapat diisi jumlah substring yang ingin diganti.

```
string = "Ayo belajar Coding di Dicoding karena Codin
g adalah bahasa masa depan"
print(string.replace("Coding", "Pemrograman", 1))
```

Output:

```
Ayo belajar Pemrograman di Dicoding karena Coding adalah bahasa masa depan
```

String Literals

Umumnya, string ditulis dengan mudah di Python, diapit oleh tanda petik tunggal. Tetapi, dalam kondisi tertentu, dibutuhkan petik tunggal di tengah string (misalnya struktur kepemilikan dalam Bahasa Inggris - That's Alice's Cat). Apabila kita menuliskannya sebagai 'That is Alice's cat.', maka Python akan salah mengira bahwa string berakhir di Alice, dan selebihnya merupakan kode yang invalid.
Namun Python memperbolehkan Anda menggunakan petik dua seperti Anda menggunakan petik tunggal. Dalam kasus sebelumnya, Anda cukup mengetikkan:

```
st = "That is Alice's cat."
```

Dan dalam contoh tersebut, Python mengenali bahwa petik tunggal adalah bagian tidak terpisahkan dari string tersebut. Bagaimana jika kita memerlukan kedua jenis petik dalam string tunggal? Python menyediakan escape character. Escape Character memungkinkan Anda untuk menggunakan karakter yang sebelumnya tidak bisa dimasukkan dalam string. Umumnya diawali dengan backslash (\) dan diikuti karakter tertentu yang diinginkan. Contohnya, untuk petik tunggal Anda dapat menambahkan seperti: \'.
Cara ini merupakan cara paling aman untuk melakukan penambahan atau penyuntingan dalam variabel. Contohnya sebagai berikut:

```
st = 'Say hi to Bob\'s mother.'
```

Python mengetahui bahwa pada Bob\'s, sebelum petik terdapat backslash (\) yang menandakan petik tunggal merupakan bagian dari string dan bukan merupakan akhir dari string. Escape character \' dan \" memungkinkan Anda untuk memasukkan karakter ' dan '' dalam bagian string. Beberapa contoh Escape Character

- \' Single quote
- \" Double quote
- \t Tab
- \n Newline (line break)
- \\ Backslash

Masukkan contoh berikut pada shell python atau notebook:

```
. print("Hello there!\nHow are you?\nI\'m doing fine.")
```

Output:

```
Hello there!
How are you?
I'm doing fine.
```

Selain tanda kutip dan kutip-dua, untuk penulisan String di Python juga bisa menggunakan 3 kutip-satu atau 3 kutip-dua, yang juga memiliki kemampuan untuk menyimpan String lebih dari satu baris (multi-line).

```
. multi_line = """Hello there!
. How are you?
. I'm fine."""
. print(multi_line)
```

Output:

```
Hello there!
How are you?
I'm fine.
```

Raw Strings

Sebaliknya, Python juga menyediakan cara untuk memasukkan string sesuai dengan apapun input atau teks yang diberikan. Metode ini dinamakan Raw Strings. Umumnya digunakan untuk regex atau beberapa implementasi lain yang sangat bergantung pada keberadaan backslash. Untuk menjadikan raw string, tambahkan huruf r sebelum pembuka string:

```
. print(r'That is Carol\'s cat.')
```

Output:

```
That is Carol\'s cat.
```

Prosedur Praktikum

Tipe Data

Dasar dari mempelajari Bahasa Pemrograman yang baru adalah pemahaman terhadap tipe data. Di sini Anda akan diajarkan tentang tipe data bawaan yang ada di Python 3 beserta contoh penggunaannya.

Numbers

Tipe numerik pada Python dibagi menjadi 3: int, float, complex. Cobalah bermain-main dengan contoh berikut:

```
a = 10
print(a, "bertipe", type(a))
b = 1.7
print(a, "bertipe", type(b))
c = 1+3j
print(c, " Bertipe bilangan kompleks? ", isinstance(1+3j,complex))
```

Output seharusnya:

```
10 bertipe <class 'int'>
1.7 bertipe <class 'float'>
(1+2j) Bertipe bilangan kompleks? True
```

Integer tidak dibatasi oleh angka atau panjang tertentu, namun dibatasi oleh memori yang tersedia. Sehingga Anda tidak perlu menggunakan variabel yang menampung big number misalnya long long (C/C++), biginteger, atau sejenisnya. Contoh kode untuk menunjukkan bahwa Python tidak membatasi output integer adalah pencarian bilangan ke-10.000 pada deret fibonacci (catatan: bilangan ke-10.000 pada deret fibonacci memiliki panjang 2.090 digit) sebagai berikut:

```
x=[0]*10005;            #inisialisasi array 0 sebanyak 10005; x[0]=
0
x[1]=1;                 #x[1]=1

for j in range(2,10001):
     x[j]=x[j-1]+x[j-2]  # Fibonacci
print(x[10000])
```

Output:

```
336447648764317832666216120051075433103021484606800639065647699746
800814421666623681555955136337340255820653326808361593737347904838
652682630408924630564318873545443695598274916066020998841839338646
527313000888302692356736131351175792974378544137521305205043477016
022647583189065278908551543661595829872796829875106312005754287834
532155151038708182989697916131278562650331954871402142875326981879
620469360978799003509623022910263681314931952756302278376284415403
605844025721143349611800230912082870460889239623288354615057765832
712525460935911282039252853934346209042452489294039017062338889910
858410651831733604374707379085526317643257339937128719375877468974
799263058370657428301616374089691784263786242128352581128205163702
980893320999057079200643674262023897831114700540749984592503606335
609338838319233867830561364353518921332797329081337326426526339897
639227234078829281779535805709936910491754708089318410561463223382
174656373212482263830921032977016480547262438423748624114530938122
065649140327510866433945175121615265453613331113140424368548051067
658434935238369596534280717687753283482343455573667197313927462736
291082106792807847180353291311767789246590899386354593278945237776
744061922403376386740040213303432974969020283281459334188268176838
930720036347956231171031012919531697946076327375892535307725523759
437884345040677155557790564504430166401194625809722167297586150269
684431469520346149322911059706762432685159928347098912847067408620
085871350162603120719031720860940812983215810772820763531866246112
782455372085323653057759564300725177443150515396009051686032203491
632226408852488524331580515348496224348482993809050704834824493274
537326245677558790891871908036620580095947431500524025327097469953
187707243768259074199396322659841474981936092852239450397071654431
564213281576889080587831834049174345562705202235648464951961124602
683139709750693826487066132645076650746115126775227486215986425307
112984411826226610571635150692600298617049454250474913781151541399
415506712562711971332527636319396069028956502882686083622410820505
624307017949761711212330660733100599473668 75
```

Batasan akurasi variabel bertipe float

Python melakukan pemotongan pada digit ke 16 pada variabel float. Float atau bilangan pecahan dibatasi akurasinya pada 15 desimal. Yang membedakan Integer dan Float adalah titik (decimal points). Misalnya dalam penulisan angka 1 jenisnya Integer, tapi jika dituliskan sebagai 1.0 artinya berjenis Float atau pecahan.

```
b = 0.1234567890123456789
print(b)
```

Output:

```
0.12345678901234568
```

Contoh jika berupa integer:

```
a = 1234567890123456789
print(a)
```

Output:

```
1234567890123456789
```

Karena Python banyak digunakan juga oleh matematikawan, tipe bilangan di Python juga mendukung bilangan imajiner dan bilangan kompleks. Nilai bilangan kompleks (complex) dituliskan dalam formulasi x + yj, yakni bagian x adalah bilangan real dan y adalah bilangan imajiner. Contohnya adalah sebagai berikut:

```
c = 1+5j
print(c)
```

Output:

```
(1+5j)
```

Operator dan Ekspresi (Expressions)

Setiap logika komputasi yang berada dalam program Anda akan menggunakan ekspresi. Sebuah ekspresi dapat terdiri dari operator dan operan. Salah satu contoh termudah adalah a + b atau 2 + 3, yang dapat kita pecah yakni + sebagai operator dan a, b, 2, atau 3 sebagai variabel/operand. Operator jelas memiliki fungsinya masing-masing dan direpresentasikan dengan simbol atau keyword tertentu.
Tip: Anda dapat melakukan expression tanpa menggunakan variabel secara langsung pada mode interaktif Python:

```
print(2 + 3)
```

Output:

```
5
```

```
print(3 * 5)
```

Output:

```
15
```

Jenis-jenis operator

Matematika dan string

+ (tambah)

Menambahkan dua objek.
3 + 5 menghasilkan 8
'a' + 'b' menghasilkan 'ab'.

- (kurang)

Mengurangkan operand kedua dari operand pertama. Jika hanya satu operand, diasumsikan nilai operand pertama adalah 0.
-5.2 adalah expression yang sama dengan 0 - 5.2 menghasilkan -5.2.
50 - 24 menghasilkan 26.
Tidak berlaku untuk string, akan menghasilkan error unsupported operand.

* (perkalian)

Mengembalikan hasil perkalian angka atau mengembalikan string yang diulang sejumlah tertentu.
2 * 3 menghasilkan 6.
'la' * 3 menghasilkan 'lalala'.

** (pangkat)

Mengembalikan operand pertama pangkat operand kedua.
3 ** 4 menghasilkan 81 (sama dengan 3 * 3 * 3 * 3).

| Tips: untuk akar dua, gunakan pangkat 0.5.

/ (pembagian)

Mengembalikan hasil pembagian operand pertama dengan operand kedua (float).
13 / 3 menghasilkan 4.333333333333333.
// (pembagian habis dibagi / div)
Mengembalikan hasil pembagian operand pertama dengan operand kedua (bilangan bulat), kecuali jika salah satu operand adalah float, akan menghasilkan float.
13 // 3 menghasilkan 4.
-13 // 3 menghasilkan -5.
9//1.81 menghasilkan 4.0.

% (modulo)

Mengembalikan sisa bagi.
13 % 3 menghasilkan 1.
-25.5 % 2.25 menghasilkan 1.5.

Operasi Bit

<< (left shift)

Menggeser representasi bit/binary dari operand pertama sebanyak operand kedua ke kiri.
2 << 2 menghasilkan 8.
2 direpresentasikan sebagai 10 dalam binary.
Geser ke kiri sebanyak 2x, menjadi 1000 (tambahkan 0 di belakangnya).
1000 dalam binary bernilai 8 dalam desimal.

>> (right shift)

Menggeser representasi bit/binary dari operand pertama sebanyak operand kedua ke kanan.
11 >> 1 menghasilkan 5.
11 direpresentasikan sebagai 1011 dalam binary.
Geser ke kanan sebanyak 1x, menjadi 101.
101 dalam binary bernilai 5 dalam desimal.

& (bit-wise AND)

Menjalankan operasi binary AND pada representasi operand pertama dan kedua.
5 & 3 menghasilkan 1.
Representasi binary 5 adalah 101 dan representasi binary 3 adalah 011.
101 and 011 bernilai 001.
001 dalam desimal adalah 1.

| (bit-wise OR)

Menjalankan operasi binary OR pada representasi operand pertama dan kedua.
5 | 3 menghasilkan 7.
Representasi binary 5 adalah 101 dan representasi binary 3 adalah 011.
101 or 011 bernilai 111.
111 dalam desimal adalah 7.

^ (bit-wise XOR)

Menjalankan operasi binary XOR pada representasi operand pertama dan kedua.
5 ^ 3 menghasilkan 6.
Representasi binary 5 adalah 101 dan representasi binary 3 adalah 011.
101 xor 011 bernilai 110.
110 dalam desimal adalah 6.

~ (bit-wise invert)

Menjalankan operasi binary invert pada representasi operand.
Nilai invert dari x adalah -(x+1), menggunakan metode Two's Complement
~5 menghasilkan -6.
Lebih lanjut mengenai Two's Complement dapat dibaca
pada https://en.wikipedia.org/wiki/Two%27s_complement

Perbandingan

< atau operator.lt (less than)

Menjalankan perbandingan apakah operand pertama lebih kecil dari operand kedua.
5 < 3 menghasilkan False and 3 < 5 menghasilkan True.
Perbandingan dapat berisi lebih dari dua operand, misalnya 3 < 5 < 7 menghasilkan True.

> atau operator.gt (greater than)

Menjalankan perbandingan apakah operand pertama lebih besar dari operand kedua.
5 > 3 menghasilkan True.

<= atau operator.le (less than or equal to)

Menjalankan perbandingan apakah operand pertama lebih kecil atau sama dengan operand kedua.
x = 3; y = 6;
x <= y menghasilkan True.

>= atau operator.ge (greater than or equal to)

Menjalankan perbandingan apakah operand pertama lebih besar atau sama dengan operand kedua.
x = 4; y = 3;
x >= y menghasilkan True.

== atau operator.eq (equal to)

Menjalankan perbandingan apakah operand pertama sama dengan operand kedua.
x = 2; y = 2;
x == y menghasilkan True.
x = 'str'; y = 'stR';
x == y menghasilkan False.
x = 'str'; y = 'str';
x == y menghasilkan True.

!= atau operator.ne (not equal to)

Menjalankan perbandingan apakah operand pertama tidak sama dengan operand kedua.
x = 2; y = 3;
x != y returns True.

Penggunaan le, lt, gt, ge, eq, ne

Mari kita implementasikan operator perbandingan berikut pada kasus jumlah kelereng berwarna hijau dan kuning.

```
from operator import *
hijau = 5
kuning = 10
print('Kelereng Hijau = {}'.format(hijau))
print('Kelereng Kuning = {}'.format(kuning))
for func in (lt, le, eq, ne, ge, gt):
    print('{}(hijau, kuning): {}'.format(func.__name__, func(hijau, kuning)))
```

Output:

```
Kelereng Hijau = 5
Kelereng Kuning = 10
lt(hijau, kuning): True
le(hijau, kuning): True
eq(hijau, kuning): False
ne(hijau, kuning): True
ge(hijau, kuning): False
gt(hijau, kuning): False
```

Boolean Operator

not (boolean NOT)

Jika x bernilai True, fungsi akan mengembalikan nilai False.
Jika x bernilai False, fungsi akan mengembalikan nilai True.
x = True;
not x akan mengembalikan nilai False.

and (boolean AND)

x AND y akan mengembalikan nilai False jika x bernilai False, atau fungsi akan mengembalikan nilai y.
x = False; y = True;
x AND y, Fungsi akan mengembalikan nilai False karena x bernilai False.
Dalam kasus ini, Python tidak akan mengevaluasi nilai y karena apapun nilai y tidak akan mempengaruhi hasil. Hal ini dinamakan short-circuit evaluation.

or (boolean OR)

x OR y, Jika x bernilai True, fungsi akan mengembalikan nilai True, atau fungsi akan mengembalikan nilai dari y.
x = True; y = False;
x OR y, fungsi akan mengembalikan nilai True.
Dalam kasus ini, Python juga menggunakan short-circuit evaluation karena apapun nilai y tidak akan mempengaruhi hasil.

Cara singkat menuliskan operasi

Jika Anda melakukan assignment kembali hasil sebuah expression, beberapa di antaranya bisa disingkat sebagai berikut:

```
a = 2
a = a * 3
```

Dapat dituliskan sebagai

```
a = 2
a *= 3
```

Perhatikan formatnya menjadi [operand] [operasi] = expression.

Urutan matematis dalam melakukan evaluasi

Jika Anda memiliki expression 2 + 3 * 4, Apakah penambahan dilakukan terlebih dahulu sebelum perkalian? Jika merujuk pada aturan yang benar, maka perkalian dilakukan lebih dahulu, sehingga perkalian memiliki urutan lebih awal/tinggi.
Berikut adalah tabel urutan yang diambil dari referensi Dokumentasi Python:

Operator	Description
lambda	Lambda expression
if - else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation
await x	Await expression
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...},{expressions...}	Binding or tuple display, list display, dictionary display, set display

Operator di Python juga bersifat asosiatif dari kiri ke-kanan. Artinya, semua operator yang memiliki tingkatan yang sama akan dijalankan berurutan dari kiri ke kanan.

Penanganan Kesalahan (Error and Exception Handling)

Ada setidaknya dua jenis kesalahan berdasarkan kejadiannya:

1. Kesalahan sintaksis (syntax errors) atau sering disebut kesalahan penguraian (parsing errors).
2. Pengecualian (exceptions) atau sering disebut kesalahan saat beroperasi (runtime errors).

Kesalahan sintaksis terjadi ketika Python tidak dapat mengerti apa yang Anda perintahkan. Sedangkan pengecualian (kesalahan saat beroperasi) terjadi ketika Python mengerti apa yang Anda perintahkan tetapi mendapatkan masalah saat mengikuti yang Anda perintahkan (terjadi saat aplikasi sudah mulai beroperasi).

Kesalahan Sintaksis

Kesalahan sintaksis biasanya sering terjadi saat Anda masih baru memulai belajar Python, misalnya contoh berikut adalah penempatan indentasi (spasi di awal) yang tidak sesuai.

```
.   print('salah indentasi')
.    File "<stdin>", line 1
.      print('salah indentasi')
.      ^
.   IndentationError: unexpected indent
```

Contoh berikut ini menampilkan kesalahan sintaksis, dimana setelah kondisi dari perintah while diharuskan ada tanda titik dua (:).

```
.   while True print('Hello world')
.    File "<stdin>", line 1
.      while True print('Hello world')
.                     ^
.   SyntaxError: invalid syntax
```

Pada kesalahan sintaksis, baris dimana kesalahan terdeteksi dimunculkan kembali, kemudian terdapat tanda panah yang menunjukkan titik paling awal dari kesalahan.
Kedua contoh di atas memiliki kelompok (tipe) kesalahan yang berbeda, yang pertama adalah IndentationError dan yang kedua adalah SyntaxError. Kemudian setelah penyebutannya, ada pesan detail kesalahan (keterangan), misalnya indentasi yang tidak diharapkan (unexpected).
Jika Anda menggunakan mode pemanggilan skrip, nama file skrip dan nomor baris dimana terjadi kesalahan akan dimunculkan. Sedangkan untuk mode interaktif pada dua contoh di atas, nama file muncul sebagai "<stdin>". Berikut adalah contoh pada pemanggilan skrip bernama contoh_salah_sintaksis.py dimana terjadi kesalahan pada baris 2.

```
.   python contoh_salah_sintaksis.py
.    File "contoh_salah_sintaksis.py", line 2
.      if True print('salah sintaksis')
.                 ^
.   SyntaxError: invalid syntax
```

Pengecualian

Meski pernyataan atau ekspresi dari Python sudah Anda tulis dengan benar, ada kemungkinan terjadi kesalahan ketika perintah tersebut dieksekusi. Kesalahan yang terjadi saat proses sedang berlangsung disebut pengecualian (exceptions) dan akan berakibat fatal jika tidak ditangani. Kebanyakan pengecualian di Python tidak ditangani oleh aplikasi, sehingga aplikasi terhenti kemudian muncul pesan kesalahan seperti contoh berikut.

```
. print(angka)
. Traceback (most recent call last):
.  File "<stdin>", line 1, in <module>
. NameError: name 'angka' is not defined
```

Misalkan Anda lupa memberikan nilai pada variabel angka, tetapi Anda langsung memanggil variabel tersebut. Secara sintaksis sudah sesuai, tapi muncul pengecualian dengan kelompok (tipe) kesalahan NameError dan pesan detail kesalahan yang menyatakan bahwa variabel angka tidak terdefinisi.
Contoh lain terkait pengecualian yang sering juga terjadi adalah operasi dari variabel yang jenisnya tidak sesuai, misalnya contoh berikut.

```
. bukan_angka = '1'
. bukan_angka + 2
. Traceback (most recent call last):
.  File "<stdin>", line 1, in <module>
. TypeError: can only concatenate str (not "int") to str
```

Pada contoh tersebut, variabel bukan_angka berjenis string, sehingga saat mengoperasikan variabel tersebut dengan angka (berjenis integer), meskipun secara sintaksis sudah sesuai, muncul pengecualian dengan kelompok (tipe) kesalahan TypeError dan pesan detail kesalahan yang menyatakan bahwa operasi penambahan untuk string (contatetation) hanya bisa dilakukan jika kedua operannya adalah string (dan bukan integer).
Seperti terlihat bahwa pada saat terjadi pengecualian, informasi yang muncul seperti saat terjadi kesalahan (errors), termasuk juga informasi nama file dan nomor baris dimana kesalahan terjadi.
Untuk mengetahui berbagai jenis pengecualian bawaan dari Python, bisa kunjungi situs dokumentasi https://docs.python.org/id/3.8/library/exceptions.html.

Penanganan Pengecualian

Pada aplikasi Python yang Anda buat bisa dilengkapi dengan penanganan terhadap pengecualian (exceptions handling) dari kelompok (tipe) kesalahan yang Anda tentukan.

Proses penanganan pengecualian menggunakan pernyataan try yang berpasangan dengan except.
Misalnya kita ingin menangani pengecualian yang terjadi jika ada pembagian angka dengan nilai nol (0).

```
>>> z = 0
>>> 1 / z

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> try:
...     x = 1 / z
...     print(x)
... except ZeroDivisionError:
...     print('tidak bisa membagi angka dengan nilai nol')

tidak bisa membagi angka dengan nilai nol
```

Perhatikan bahwa operasi aplikasi berhenti di x = 1 / z, sedangkan bagian print(x) tidak sempat dioperasikan, karena aplikasi sudah mengalami pengecualian, sehingga yang tercetak adalah operasi print('tidak bisa membagi angka dengan nilai nol').
Pada operasi yang dicontohkan di atas, penanganan pengecualian untuk ZeroDivisionError dilakukan sehingga aplikasi tidak lagi keluar dari eksekusi karena kesalahan, tapi digantikan dengan mencetak pesan ke layar. Pada contoh ini kita fokus pada penanganan pengecualian, meskipun ada cara lain untuk menyelesaikannya, misal menggunakan kondisi (percabangan) untuk menghindari nilai nol.
Pernyataan except dilanjutkan dengan kelompok (tipe) kesalahan yang ingin ditangani, atau bisa juga berupa tuple dari satu atau lebih tipe kesalahan yang akan ditangani. Di contoh berikut, menangani FileNotFoundError sebagai tuple satu elemen, jangan lupa dalam menuliskan tuple satu elemen harus tetap diakhiri dengan koma.

```
>>> try:
...     with open('contoh_tidak_ada.py') as file:
...         print(file.read())
... except (FileNotFoundError, ):
...     print('file tidak ditemukan')
...
file tidak ditemukan
```

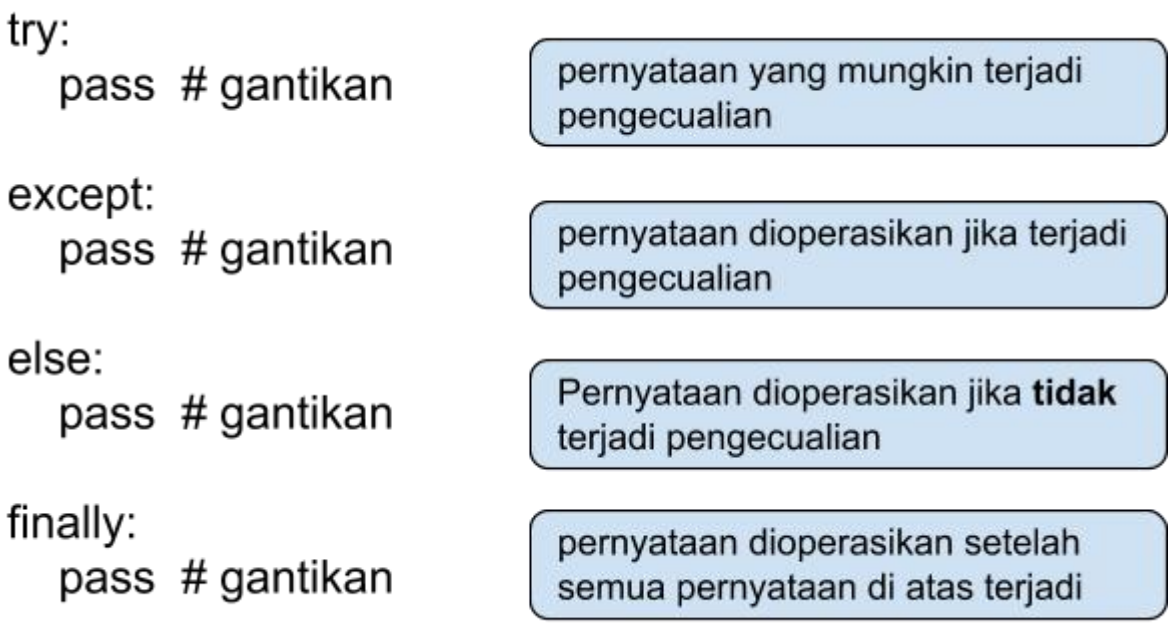
Pada operasi di atas, aplikasi akan membuka dan mengakses file bernama contoh_tidak_ada.py, tetapi file tersebut tidak ada di direktori dimana aplikasi Python tersebut berada, selanjutnya akan terjadi pengecualian (exceptions) tetapi ditangani, dalam pasangan pernyataan try dan except, sehingga aplikasi tidak terhenti tetapi tercetak di layar bahwa file tidak ditemukan.
Dalam aplikasi yang lebih kompleks, penanganan pengecualian dapat menggunakan pernyataan except lebih dari satu. Di contoh berikutnya akan menggunakan pernyataan except

lebih dari satu (untuk satu pernyataan try), maupun menggunakan satu pernyataan except yang menangani lebih dari satu tipe kesalahan yang digabung dalam sebuah tuple.

```
>>> d = {'ratarata': '10.0'}
>>> try:
...     print('rata-rata: {}'.format(d['rata_rata']))
... except KeyError:
...     print('kunci tidak ditemukan di dictionary')
... except ValueError:
...     print('nilai tidak sesuai')
...
kunci tidak ditemukan di dictionary
>>> try:
...     print('rata-rata: {}'.format(d['ratarata']/3))
... except KeyError:
...     print('kunci tidak ditemukan di dictionary')
... except (ValueError, TypeError):
...     print('nilai atau tipe tidak sesuai')
...
nilai atau tipe tidak sesuai
>>> try:
...     print('pembulatan rata-rata: {}'.format(int(d['ratarata
']))))
... except (ValueError, TypeError) as e:
...     print('penangan kesalahan: {}'.format(e))
...
penangan kesalahan: invalid literal for int() with base 10: '10.0'
```

Pada contoh tersebut, yang paling awal terjadi pengecualian untuk tipe kesalahan KeyError karena dalam dictionary d tidak memiliki kunci (key) rata_rata, yang ada adalah kunci ratarata. Kemudian contoh selanjutnya terjadi pengecualian untuk tipe kesalahan TypeError karena nilai d[’ratarata’] memiliki tipe string, sehingga tidak dapat dibagi dengan integer (angka) 3. Dalam penanganan kesalahannya, satu buah pernyataan except menangani tipe kesalahan ValueError atau TypeError, sehingga cocok salah satunya akan menampilkan ke layar bahwa nilai atau tipe tidak sesuai.
Di bagian paling akhir contoh, terjadi pengecualian untuk tipe kesalahan ValueError karena berusaha melakukan konversi (casting) dari sebuah string ke integer dengan format yang tidak sesuai (bilangan bulat seharusnya tidak memiliki titik dalam penulisannya). Dalam penulisan penanganan kesalahannya digunakan variasi lain untuk mendapatkan pesan kesalahan sebagai variabel e untuk kemudian variabel tersebut dicetak dalam pesan yang ditampilkan ke layar.

Bentuk lengkap dari pernyataan try dapat dilihat pada bagan berikut, terdiri dari pernyataan except, else, finally.



Menghasilkan Pengecualian

Dalam membuat aplikasi, ada kemungkinan Anda butuh untuk menghasilkan pengecualian (raise exceptions), salah satu caranya bisa dengan menggunakan pengecualian yang sudah ada, hanya ditambahkan informasi detailnya saja.
Misalnya dalam contoh berikut, Anda mewajibkan sebuah dictionary memiliki kunci (key) total.

```
>>> d = {'ratarata': '10.0'}
>>> if 'total' not in d:
...     raise KeyError('harus memiliki total')
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyError: 'harus memiliki total'
```

Percabangan

Beberapa contoh aplikasi dasar untuk pengenalan percabangan yang sering digunakan adalah:

- Genap atau Ganjil.
- Positif atau Negatif.
- Positif, Negatif, atau Nol.

If

Seperti bahasa pemrograman lainnya, Python juga memiliki statemen percabangan IF. Di Python, expression diletakkan setelah if, dan keputusan ditentukan berdasarkan nilai kebenaran dari expression tersebut.
Tip: Python menganggap setiap nilai non-zero dan non-null sebagai True dan nilai zero/null sebagai False.
Jika expression dievaluasi sebagai True, maka blok statement di dalam if statement akan dieksekusi. Sesuai konvensi, blok ini memiliki indentasi masuk setelah tanda titik dua (:). Jika expression dievaluasi sebagai False, maka blok selanjutnya (setelah statement IF) yang akan dijalankan. Contoh:

```
kelerengku = 10
if kelerengku:
    print ("Cetak ini jika benar")
    print (kelerengku)
```

Output:

```
Cetak ini jika benar
10
```

Anda dapat menyingkat penulisan statement yang akan dieksekusi jika ia terwakili dalam 1 baris. Misalnya:

```
if kerelengku: hitung()
```

Else

Statement Else dapat dikombinasikan dengan IF Statement, sebagai jalan keluar saat kondisi / hasil evaluasi bernilai False. Else bersifat opsional dan tunggal. Mari kita implementasikan dalam kasus pengecekan tinggi badan pengunjung di suatu wahana.

```
tinggi_badan = int(input("Masukkan tinggi badan Anda : "))
if tinggi_badan>=160:
    print ("Silakan, Anda boleh masuk")
else:
    print ("Maaf, Anda belum boleh masuk")
```

Output 1:

```
Masukkan tinggi badan Anda : 160
Silakan, Anda boleh masuk
```

Output 2:

```
Masukkan tinggi badan Anda : 140
Maaf, Anda belum boleh masuk
```

Mari implementasikan pada kasus yang berbeda, kali ini kasusnya adalah pengecekan bilangan ganjil atau genap pada suatu variabel.

```
bilangan = 4
if bilangan % 2 == 0:
    print('Bilangan {} adalah genap'.format(bilangan))
else:
    print('Bilangan {} adalah ganjil'.format(bilangan))
```

Output:

```
Bilangan 4 adalah genap
```

Elif - Alternatif untuk Switch/Case dan IF bertingkat di python

Elif adalah kependekan dari else if, dan merupakan alternatif untuk if bertingkat atau switch/case di beberapa bahasa pemrograman lain. Sebuah IF Statement dapat diikuti satu atau lebih statement elif (opsional & tidak dibatasi). Mari kita implementasikan pada kasus penilaian tugas siswa.

```
nilai = int(input("Masukkan nilai tugas Anda : "))
if nilai>80:
    print("Selamat! Anda mendapat nilai A")
    print("Pertahankan!")
elif nilai>70:
    print("Hore! Anda mendapat nilai B")
    print("Tingkatkan!")
elif nilai>60:
    print("Hmm.. Anda mendapat nilai C")
    print("Ayo semangat!")
else:
    print("Waduh, Anda mendapat nilai D")
    print("Yuk belajar lebih giat lagi!")
```

Output 1:

```
Masukkan nilai tugas Anda : 85
Selamat! Anda mendapat nilai A
Pertahankan!
```

Output 2:

```
Masukkan nilai tugas Anda : 75
Hore! Anda mendapat nilai B
Tingkatkan!
```

Output 3:

```
Masukkan nilai tugas Anda : 65
Hmm.. Anda mendapat nilai C
Ayo semangat!
```

Output 4:

```
Masukkan nilai tugas Anda : 30
Waduh, Anda mendapat nilai D
Yuk belajar lebih giat lagi!
```

Catatan: Jika sudah memenuhi salah satu kondisi if/elif, maka program akan keluar dari blok IF Statement. Anda harus memastikan urutan secara logika, IF, Elif, dan Else dalam tingkatan yang tepat. Contoh yang perlu diperhatikan adalah sebagai berikut:

Input Nilai	#Kasus 1
	
	if nilai>80:
	print("Selamat! Anda mendapat nilai A")
	print("Pertahankan!")
	elif nilai>70:
	print("Hore! Anda mendapat nilai B")
	print("Tingkatkan!")
	elif nilai>60:
	print("Hmm.. Anda mendapat nilai C")
	print("Ayo semangat!")
	else:
	print("Waduh, Anda mendapat nilai D")
	print("Yuk belajar lebih giat lagi!")
		#Kasus 2
		
		if nilai>0:
		print("Selamat! Anda mendapat nilai A")
		print("Pertahankan!")
		elif nilai<80:
		print("Hore! Anda mendapat nilai B")
		print("Tingkatkan!")
		elif nilai<70:
		print("Hmm.. Anda mendapat nilai C")
		print("Ayo semangat!")
		else:
		print("Waduh, Anda mendapat nilai D")
		print("Yuk belajar lebih giat lagi!")

85	Masukkan nilai tugas Anda: 85 Selamat! Anda mendapat nilai A Pertahankan!	Masukkan nilai tugas Anda: 85 Selamat! Anda mendapat nilai A Pertahankan!
65	Masukkan nilai tugas Anda: 65 Hmm.. Anda mendapat nilai C Ayo semangat!	Masukkan nilai tugas Anda: 65 Selamat! Anda mendapat nilai A Pertahankan!
30	Masukkan nilai tugas Anda: 30 Waduh, Anda mendapat nilai D Yuk belajar lebih giat lagi!	Masukkan nilai tugas Anda: 30 Selamat! Anda mendapat nilai A Pertahankan!

Pada #Case 2, elif dan else tidak pernah dijalankan karena nilai berapapun (yang bernilai positif) akan selalu masuk pada IF (klausa pertama).
Mari kita implementasikan pada kasus pengecekan bilangan positif, negatif, atau nol.

```
.   bilangan = -3
.   if bilangan > 0:
.       print('Bilangan {} adalah positif'.format(bilangan))
.   elif bilangan < 0:
.       print('Bilangan {} adalah negatif'.format(bilangan))
.   else:
.       print('Bilangan {} adalah nol'.format(bilangan))
```

Output:

```
Bilangan -3 adalah negatif
```

Ternary Operators

Ternary operator lebih dikenal sebagai conditional expressions pada Python. Operator menentukan sesuatu berdasarkan kondisi True atau False. Jika statement atau klausa if Anda cukup sederhana, maka ternary Operators akan sangat membantu.

IF	Ternary

if (condition):
condition_if_true
else:
condition_if_false
	condition_if_true if condition else condition_if_false

lulus = True
if (lulus):
kata = ” selamat”
else:
kata = “perbaiki”
	lulus = True
	kata = "selamat" if lulus else "perbaiki"

Perbandingan klausa IF dengan ternary Operators:

IF	Ternary_Tuples

if (condition):
condition_if_true
else:
condition_if_false
	(condition_if_false, condition_if_true)[condition]

lulus = True
if (lulus):
kata=” selamat”
else:
kata= “perbaiki”
	nice = True
	kata= ("perbaiki", "selamat")[lulus]

Opsi lain dari ternary operators melibatkan tuples. Contoh kodenya berikut:
Pada tuple ini, dimanfaatkan nilai [0] sebagai False dan [1] sebagai True.
Aplikasi kedua ini menurut beberapa aktivis kurang ‘pythonic’, salah satunya karena cukup membingungkan untuk meletakkan klausa saat True atau False. Selain itu, kedua nilai akan tetap dievaluasi walaupun hanya dibutuhkan salah satunya. Lihat contoh berikut:

```
kondisi = True
print(2 if kondisi else 1/0)
#Output is 2

print((1/0, 2)[kondisi])
#Eror Pembagian Nol akan muncul
```

For

Seperti di bahasa pemrograman lainnya, Python juga memiliki fungsi for. Bedanya di Python, For tidak hanya untuk perulangan dengan jumlah finite (terbatas), melainkan lebih ke fungsi yang dapat melakukan perulangan pada setiap jenis variabel berupa kumpulan atau urutan. Variabel yang dimaksud bisa berupa list, string, ataupun range. Jika sebuah list atau urutan berisi expression, maka Ia akan dievaluasi terlebih dahulu. Kemudian item pertama pada urutan/list akan diassign sebagai variabel iterating_var. Setelahnya, blok statement akan dieksekusi, berlanjut ke item berikutnya, berulang, hingga seluruh urutan habis.

```
.   for huruf in 'Dicoding':  # Contoh pertama
.       print('Huruf: {}'.format(huruf))
.
.   flowers = ['mawar', 'melati', 'anggrek']
.   for flower in flowers:  # Contoh kedua
.       print('Flower: {}'.format(flower))
```

Output:

```
Huruf : D
Huruf : i
Huruf : c
Huruf : o
Huruf : d
Huruf : i
Huruf : n
Huruf : g
Flower: mawar
Flower: melati
Flower: anggrek
```

Anda juga dapat melakukan perulangan berdasarkan indeks atau range dengan memanfaatkan fungsi len():

```
.   flowers = ['mawar', 'melati', 'anggrek']
.   for index in range(len(flowers)):
.       print('Flowers: {}'.format(flowers[index]))
```

Output:

```
Flower : mawar
Flower : melati
Flower   : anggrek
```

While

While pada bahasa Python digunakan untuk mengeksekusi statement selama kondisi yang diberikan terpenuhi (True). Kondisi dapat berupa expression apapun, dan harap diingat bahwa True di Python termasuk semua nilai non-zero. Saat kondisi menjadi False, program akan melanjutkan ke baris setelah blok statement.
Seperti for dan semua statement percabangan, blok statement yang mengikuti kondisi while dan memiliki posisi indentasi yang sama, dianggap blok statement yang akan dieksekusi.

Contoh:

```
.   count = 0
.   while (count < 7):
.       print('Hitungannya adalah: {}'.format(count))
.       count = count + 1
```

Output:

```
Hitungannya adalah: 0
Hitungannya adalah: 1
Hitungannya adalah: 2
Hitungannya adalah: 3
Hitungannya adalah: 4
Hitungannya adalah: 5
Hitungannya adalah: 6
```

Seperti pada bahasa lainnya, eksekusi statement while mungkin bersifat infinit / infinite loop saat sebuah kondisi tidak pernah bernilai False. Contohnya sebagai berikut:

```
.   var = 1
.   while var == 1:  # This constructs an infinite loop
.       num = input('Masukkan angka: ')
.       print('Anda memasukkan angka: {}'.format(num))
.
.
.   while True:  # This constructs an infinite loop
.       num = input('Masukkan angka: ')
.       print('Anda memasukkan angka: {}'.format(num))
```

Potongan kode di atas tidak akan pernah bernilai False karena nilai var tidak pernah berubah. Untuk menghentikan infinite loop, gunakan CTRL (atau CMD⌘) - C untuk menghentikannya dan keluar dari program.
Anda juga dapat menyingkat penulisan blok statement While jika statement Anda cukup terwakili oleh satu baris.

```
.   while (var1): do_something()
```

Perulangan Bertingkat

Ada kalanya Anda perlu untuk melakukan perulangan bertingkat, misalnya untuk menghasilkan contoh print-out berikut:

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

Anda dapat melakukannya dengan kode berikut:

```
. for i in range(0, 6):
.     for j in range(0, 6 - i):
.         print('*', end='')
.     print()
```

Break

Pernyataan break menghentikan perulangan kemudian keluar, dilanjutkan dengan mengeksekusi pernyataan (statement) setelah blok perulangan. Salah satu penggunaannya yang paling sering adalah sebuah kondisi eksternal yang membutuhkan program untuk keluar dari perulangan. Jika Anda memiliki perulangan bertingkat, break akan menghentikan perulangan sesuai dengan tingkatan atau di perulangan mana ia berada. Namun jika ia diletakkan di perulangan dengan kedalaman kedua misalnya, hanya perulangan itu saja yang berhenti, tidak dengan perulangan utama.

Contoh 1:

```
. for huruf in 'Dico ding':
.     if huruf == ' ':
.         break
.     print('Huruf saat ini: {}'.format(huruf))
```

Output contoh 1:

```
Huruf saat ini: D
Huruf saat ini: i
Huruf saat ini: c
Huruf saat ini: o
```

Contoh 2, cetak bintang dengan memanfaatkan fungsi break

```
.   for i in range (0,10):
.       for j in range (0,10):
.           if j>i:
.               print()
.               break
.           else:
.               print("*",end="")
```

Output contoh 2:

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

Continue

Pernyataan continue akan membuat iterasi saat ini berhenti, kemudian melanjutkan ke iterasi berikutnya, mengabaikan pernyataan (statement) yang berada antara continue hingga akhir blok perulangan.

Contoh 1:

```
.   for huruf in 'Dico ding':
.       if huruf == ' ':
.           continue
.       print('Huruf saat ini: {}'.format(huruf))
```

Output contoh 1:

```
Huruf saat ini: D
Huruf saat ini: i
Huruf saat ini: c
Huruf saat ini: o
Huruf saat ini: d ## perhatikan spasi dilewati
Huruf saat ini: i
Huruf saat ini: n
Huruf saat ini: g
```

Contoh 2, cetak bintang yang sama dengan contoh 2 pada pembahasan fungsi break, dengan 1 loop dan 1 if (tanpa else):

```
.   jumlahbaris = 10
.   baris = 0
.   bintang = 0
.   while baris < jumlahbaris:
.       if (bintang) >= (baris+1):
.           print()
.           baris = baris+1
.           bintang=0
.           continue     ##saat masuk ke if, maka bagian print * diluar if ti
    dak akan dijalankan, langsung ulang ke while
.       print("*",end="")
.       bintang= bintang+1
```

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

Else setelah For

Pada Python juga dikenal fungsi else setelah for. Fungsinya diutamakan pada perulangan yang bersifat pencarian - untuk memberikan jalan keluar program saat pencarian tidak ditemukan. Struktur umumnya adalah sebagai berikut:

```
.   for item in items:
.       if cari(item):
.           #ditemukan!
.           proses_item()
.           break
.   else:
.       #Item tidak ditemukan
.       not_found_in_container()
```

Anda mungkin melihat sesuatu yang menarik. Ada If dan Else, namun dalam tingkatan yang berbeda, apakah mereka terkait?

Saat sebuah perulangan dijalankan, fungsi if akan dievaluasi. Saat ia pernah sekali saja benar, maka else tidak akan dieksekusi. Dengan kata lain, if yang berada dalam perulangan harus selalu salah untuk memicu blok statemen else dijalankan. Lebih singkat lagi, struktur pseudocode yang diikuti dalam membuat else pada perulangan adalah seperti berikut:

```
.   if any(something_about(thing) for each thing in container):
.       do_something(that_thing)
.   else:
.       no_such_thing()
```

Contoh penggunaan dari for-else dapat dilihat pada potongan kode berikut (jalankan di konsol sebelum melanjutkan ke bagian selanjutnya):

```
.   for n in range(2, 10):
.       for x in range(2, n):
.           if n % x == 0:
.               print(n, 'equals', x, '*', n/x)
.               break
```

Potongan kode di atas melakukan pencarian faktor dari setiap bilangan antara 2 s/d 9. Namun demikian, pada saat bilangan n bernilai 2, 3, 5, 7, program tidak akan mencetak apapun. Ini karena bilangan-bilangan tersebut merupakan bilangan prima. Hal ini bisa diatasi dengan menambahkan keterangan else dan mencetak bahwa bilangan tersebut adalah bilangan prima:

```
.   for n in range(2, 10):
.       for x in range(2, n):
.           if n % x == 0:
.               print( n, 'equals', x, '*', n/x)
.               break
.       else:
.           # loop fell through without finding a factor
.           print(n, ' adalah bilangan prima')
```

Else setelah While

Berbeda dengan Else setelah For, pada statement while, blok statement else akan selalu dieksekusi saat kondisi pada while menjadi salah. Contoh mudahnya adalah sebagai berikut:

```
.   n = 10
.   while n > 0:
.       n = n - 1
.       if n == 7:
.           break
.       print(n)
.   else:
.       print("Loop selesai")
```

Output:

```
9
8
```

Pada contoh diatas, loop akan di break saat nilai n == 7, saat keluar dari perulangan, maka python tidak akan memunculkan tulisan Loop selesai, namun jika tidak dilakukan break (perulangan berakhir dengan normal):

```
n = 10
while n > 0:
    n = n - 1
    print(n)
else:
    print("Loop selesai")
```

Output:

```
9
8
7
6
5
4
3
2
1
0
Loop selesai
```

Pass

Digunakan jika Anda menginginkan sebuah pernyataan atau blok pernyataan (statement), namun tidak melakukan apapun - melanjutkan eksekusi sesuai dengan urutannya. Kontrol ini banyak digunakan saat Anda belum melakukan implementasi (atau menyiapkan tempat untuk implementasi), serta membiarkan program tetap berjalan saat misalnya Anda mengalami kegagalan atau exception.
Pass statement adalah operasi bersifat Null (kosong), tidak ada yang terjadi saat ia dipanggil. Contohnya:

```
def sebuahfungsi():
    pass
```

Output:

```
#tidak ada
```

Jika Anda mendeklarasi sebuah fungsi tanpa kode apapun, justru akan terjadi error:

```
def sebuahfungsi():
```

Output:

```
File "<test.py>", line 2
^
IndentationError: expected an indented block
```

Contoh pass untuk mengantisipasi exception/kegagalan fungsi, perhatikan contoh kode yang belum ditambahkan pass berikut:

```
var1=""
while(var1!="exit"):
    var1=input("Please enter an integer (type exit to exit): ")
    print(int(var1))
```

Output:

```
Please enter an integer (type exit to exit): 1
1
Please enter an integer (type exit to exit): a
Traceback (most recent call last):
File "testp3.py", line 7, in <module>
print(int(var1))
ValueError: invalid literal for int() with base 10: 'a'
```

Kita dapat membaca bahwa program gagal saat mencoba proses konversi variabel var1. Saat program mengalami kegagalan atau exception, ia akan langsung keluar. Bandingkan dengan pendekatan berikut (mengenai exception, import/library sys, dan try/except akan dibahas pada modul-modul pembelajaran berikutnya:

```
import sys
data=''
while(data!='exit'):
    try:
        data=input('Please enter an integer (type exit to exit): ')
        print('got integer: {}'.format(int(data)))
    except:
        if data == 'exit':
            pass  # exit gracefully without prompt any error
        else:
            print('error: {}'.format(sys.exc_info()[0]))
```

Outputnya saat dijalankan:

```
Please enter an integer (type exit to exit): 1
got integer: 1
Please enter an integer (type exit to exit): a
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): b
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): c
Unexpected error: <class 'ValueError'>
Please enter an integer (type exit to exit): exit
```

List

List adalah jenis kumpulan data terurut (ordered sequence), dan merupakan salah satu variabel yang sering digunakan pada Python. Serupa, namun tak sama dengan array pada bahasa pemrograman lainnya. Bedanya, elemen List pada Python tidak harus memiliki tipe data yang sama. Mendeklarasikan List cukup mudah dengan kurung siku dan elemen yang dipisahkan dengan koma.
Setiap data di dalamnya dapat diakses dengan indeks yang dimulai dari 0.

```
a = [1, 2.2, 'python']
```

Python mengenal slicing operator [] yang dapat melakukan ekstraksi sebuah item atau beberapa item yang berada dalam range tertentu pada tipe data urutan (sequences), misalnya list, string dan tuple. Beberapa tipe urutan juga mendukung "extended slicing" dengan parameter ketiga berupa "step".

- **x[0]** artinya mengambil elemen paling awal, dengan index 0 dari List x.
- **x[5]** artinya mengambil elemen dengan index 5 dari List x.
- **x[-1]** artinya mengambil elemen dengan index paling belakang ke-1 dari List x.
- **x[3:5]** artinya membuat list dari anggota elemen List x dengan index 3 hingga sebelum index 5 (tidak termasuk elemen dengan index 5, dalam hal ini hanya index 3-4).
- **x[:5]** artinya membuat list dari anggota elemen List x paling awal hingga sebelum index 5 (tidak termasuk elemen dengan index 5, dalam hal ini hanya index 0-4).
- **x[-3:]** artinya membuat list dari anggota elemen List x mulai index ke-3 dari belakang hingga paling belakang.
- **x[1:7:2]** artinya membuat list dari anggota elemen List x dengan index 1 hingga sebelum index 7, dengan "step" 2 (dalam hal ini hanya index 1, 3, 5).

```
x = [5,10,15,20,25,30,35,40]
print(x[5])
print(x[-1])
print(x[3:5])
print(x[:5])
print(x[-3:])
print(x[1:7:2])
```

Output:

```
30
40
[20, 25]
[5, 10, 15, 20, 25]
[30, 35, 40]
[10, 20, 30]
```

Elemen pada list dapat diubah atau ditambahkan. Misalnya untuk melakukan perubahan kemudian penambahan:

```
x = [1,2,3]
x[2]=4
print (x)
```

Output:

```
[1, 2, 4]
```

```
x = [1,2,3]
x[2]=4
x.append(5)
print(x)
```

Output:

```
[1, 2, 4, 5]
```

Untuk menghapus item pada list, gunakan fungsi del. Ingat bahwa Indeks Python dimulai dari 0:

```
binatang = ['kucing', 'rusa', 'badak', 'gajah']
del binatang[2]
print(binatang)
```

Output:

```
['kucing', 'rusa', 'gajah']
```

Coba tambahkan kembali:

```
del bintang [2]
print(binatang)
```

Pada baris terbawah kode di atas, maka output akan menjadi:

Output:

```
['kucing', 'rusa']
```

Slicing pada String

Karena string mirip dengan list, maka slicing operator [ ] juga dapat digunakan pada string untuk mengambil isinya atau bahkan substring. Sebuah string utuh bersifat mutable (bisa diubah), namun elemennya bersifat immutable (tidak bisa diubah).

```
s = "Hello World!"
print(s[4])     #ambil karakter kelima dari string s
print(s[6:11])      #ambil karakter ketujuh hingga sebelas dari str
ing s
s[5]="d"        #ubah karakter keenam dari string s menjadi "d", se
harusnya gagal karena immutable
s = "Halo Dunia!"   #ubah isi string s menjadi "Halo Dunia!", seharu
snya berhasil karena mutable
print (s)
```

Output:

```
'o'
'World'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
'Halo Dunia!'
```

Tuple

Tuple adalah jenis dari list yang tidak dapat diubah elemennya. Umumnya tuple digunakan untuk data yang bersifat sekali tulis, dan dapat dieksekusi lebih cepat. Tuple didefinisikan dengan kurung dan elemen yang dipisahkan dengan koma.

```
t = (5,'program', 1+3j)
```

Seperti list, kita dapat melakukan slicing, namun pada tuple kita tidak dapat melakukan perubahan:

```
t = (5,'program', 1+3j)
print(t[1])
print(t[0:3])
print(t[0]=10)
```

Output:

```
'program'
(5, 'program', (1+3j))
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Set

Set adalah kumpulan item bersifat unik dan tanpa urutan (unordered collection). Didefinisikan dengan kurawal dan elemennya dipisahkan dengan koma. Pada Set kita dapat melakukan union dan intersection, sekaligus otomatis melakukan penghapusan data duplikat.

```
a = {1,2,2,3,3,3}
print(a)
```

Output:

```
{1, 2, 3}
```

Karena set bersifat unordered, maka kita tidak bisa mengambil sebagian data / elemen datanya menggunakan proses slicing.

```
a = {1,2,3}
print(a[1])
```

Output:

```
Traceback (most recent call last):
File "<string>", line 301, in runcode
File "<interactive input>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Dictionary

Dictionary pada Python adalah kumpulan pasangan kunci-nilai (pair of key-value) yang bersifat tidak berurutan. Dictionary dapat digunakan untuk menyimpan data kecil hingga besar. Untuk mengakses datanya, kita harus mengetahui kuncinya (key). Pada Python, dictionary didefinisikan dengan kurawal dan tambahan definisi berikut:

1. Setiap elemen pair key-value dipisahkan dengan koma (,).
2. Key dan Value dipisahkan dengan titik dua (:).
3. Key dan Value dapat berupa tipe variabel/obyek apapun.

```
d = {1:'value','key':2}
print(type(d))
```

Output:

```
<class 'dict'>
```

```
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);
```

Output:

```
<class 'dict'>
d[1] = value
d['key'] =  2
```

Dictionary bukan termasuk dalam implementasi urutan (sequences), sehingga tidak bisa dipanggil dengan urutan indeks. Misalnya dalam contoh berikut dicoba dengan indeks 2, tetapi menghasilkan error (KeyError) karena tidak ada kunci (key) 2:

```
d = {1:'value','key':2}
print(type(d))
print("d[1] = ", d[1]);
print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

Output:

```
<class 'dict'>
d[1] = value
d['key'] = 2

---------------------------------------------------------------------------
KeyError Traceback (most recent call last)
<ipython-input-7-4b566e677ca2> in <module>()
1 d = {1:'value','key':2}
----> 2 print("d[2] = ", d[2]);

KeyError: 2
```

Konversi (conversion, cast) antar tipe data

Anda juga dapat melakukan konversi kumpulan data (set, list, tuple).

```
print(set([1,2,3]))
```

Output:

```
{1, 2, 3}
```

```
print(tuple({5,6,7}))
```

Output:

```
(5, 6, 7)
```

```
print(list('hello'))
```

Output:

```
['h', 'e', 'l', 'l', 'o']
```

Untuk konversi ke dictionary, data harus memenuhi persyaratan key-value. Berikut adalah dua contoh konversi:
List dari beberapa List yang isinya pasangan nilai menjadi Dictionary.
Serta konversi List dari beberapa Tuple yang isinya pasangan nilai menjadi Dictionary.

```
print(dict([[1,2],[3,4]]))
```

Output:

```
{1: 2, 3: 4}
```

```
print(dict([(3,26),(4,44)]))
```

Output:

```
{3: 26, 4: 44}
```

Fungsi

Di matematika, fungsi adalah proses yang merelasikan antara sebuah masukan (input) dan keluaran (output). Pada Python, selain fungsi relasi tersebut, fungsi juga adalah salah satu cara untuk mengorganisasikan kode - dengan tujuan akhir kode dapat digunakan kembali (reusability).
Beberapa syarat umum fungsi adalah modularisasi dan fungsionalitasnya. Jadi sebaiknya fungsi hanya memiliki satu kegunaan spesifik namun dapat digunakan kembali. Fungsi-fungsi umum telah disediakan oleh Python misalnya print(). Namun Anda dapat selalu mendefinisikan fungsi Anda sendiri.

Mendefinisikan fungsi

Fungsi didefinisikan dengan keyword def diikuti dengan nama fungsi dan parameternya dalam kurung ().

```
.   def nama_fungsi( parameter )
```

Secara opsional, Anda dapat menambahkan docstring - string dokumentasi yang menjelaskan konteks fungsi. Blok kode dalam setiap fungsi dimulai dengan titik dua dan menggunakan indentasi. Fungsi berhenti ketika terdapat statement return [expression] yang mengembalikan [expression] kepada pemanggilnya. Anda juga bisa membuat fungsi tidak mengembalikan keluaran dengan return None.
Sintaksis fungsi secara lengkap pada Python:

```
.   def nama_fungsi( parameter ):
.       "dokumentasi fungsi"
.       statemen atau kode fungsi
.       return [expression]
```

Secara default, Python akan memposisikan setiap parameter sesuai dengan urutan pendaftaran pada saat didefinisikan, dan harus dipanggil sesuai dengan urutan tersebut.
Contoh: fungsi berikut akan menerima sebuah string sebagai parameter dan mencetaknya.

```
.   def cetak( param1 ):
.       print(param1)
.       return
```

Sintaksis return tanpa ekspresi atau return None dapat juga tidak dituliskan. Fungsi di atas akan sama seperti di bawah ini.

```
.   def cetak( param1 ):
.       print(param1)
```

Memanggil Fungsi

Mendefinisikan sebuah fungsi hanya memberikan namanya, menentukan parameter yang ingin menjadi bagian dari fungsi dan struktur dasar kode tersebut. Setelah struktur dasar terpenuhi, Anda dapat memanggilnya pada fungsi yang lain atau dari Python prompt. Contoh berikut untuk memanggil fungsi printme().

```
def cetak( param1 ):
    print(param1)
    return

#panggil
cetak("Panggilan Pertama")
cetak("Panggilan Kedua")
```

Saat kode diatas dijalankan, akan menghasilkan keluaran berikut:
Output:

```
Panggilan Pertama
Panggilan Kedua
```

Return

Pernyataan return [expression] akan membuat eksekusi program keluar dari fungsi saat itu, sekaligus mengembalikan nilai tertentu. Nilai return yang tidak mengembalikan (ekspresi) nilai bersifat sama dengan contoh di bawah ini.

```
return None
```

Contoh fungsi dengan return:

```
def kali(angka1, angka2):
    # Kalikan kedua parameter
    hasil = angka1 * angka2
    print('Dicetak dari dalam fungsi: {}'.format(hasil))
    return hasil

# Panggil fungsi kali
keluaran = kali(10, 20);
print('Dicetak sebagai kembalian: {}'.format(keluaran))
```

Saat dipanggil:

```
Dicetak dari dalam fungsi: 200
Dicetak sebagai kembalian: 200
```

Nilai kembalian dari sebuah fungsi dapat disimpan dalam sebuah variabel. Ini yang akan membedakan sebuah fungsi yang mengembalikan nilai dengan sebuah fungsi yang tidak mengembalikan nilai (sering disebut sebagai prosedur). Cobalah kode berikut ini:

```
```

```
. def kuadrat(x):
.     return x*x
. a = 10
. k = kuadrat(a)
. print('nilai kuadrat dari {} adalah {}'.format(a, k))
```

Pass by reference vs value

Seluruh parameter (argumen) pada bahasa Python bersifat "passed by reference". Artinya saat Anda mengubah sebuah variabel, maka data yang mereferensi padanya juga akan berubah, baik di dalam fungsi, maupun di luar fungsi pemanggil. Kecuali jika anda melakukan operasi assignment yang akan mengubah reference parameter.
Contohnya:

```
. def ubah(list_saya):
.     list_saya.append([1, 2, 3, 4])
.     print('Nilai di dalam fungsi: {}'.format(list_saya))
.
. # Panggil fungsi ubah
. list_saya = [10, 20, 30]
. ubah(list_saya)
. print('Nilai di luar fungsi: {}'.format(list_saya))
```

Dapat dilihat dalam kode diatas, objek mylist yang direferensi adalah sama. Sehingga saat melakukan perubahan, maka perubahannya terjadi pada mylist baik didalam maupun diluar fungsi ubah:

```
Nilai di dalam fungsi:  [10, 20, 30, [1, 2, 3, 4]]
Nilai di luar fungsi:  [10, 20, 30, [1, 2, 3, 4]]
```

Namun Anda harus berhati-hati karena assignment variabel bernama sama dengan parameter, berarti membuat variabel baru dalam scope lokal dan tidak terkait dengan variabel global.

```
. def ubah(list_saya):
.     "Deklarasi Variabel list_saya berikut hanya dikenali (berlaku)
di dalam fungsi ubah"
.     list_saya = [1, 2, 3, 4]
.     print ('Nilai di dalam fungsi: {}'.format(list_saya))
.
. # Panggil fungsi ubah
. list_saya = [10, 20, 30]
. ubah(list_saya)
. print('Nilai di luar fungsi: {}'.format(list_saya))
```

Variabel mylist dibuat kembali versi localnya dalam fungsi ubah dengan operator assignment (sama dengan), sehingga nilai mylist akan berbeda karena bersifat lokal dalam fungsi ubah saja. Hasilnya akan sebagai berikut:

```
Nilai di dalam fungsi:  [1, 2, 3, 4]
Nilai di luar fungsi: [10, 20, 30]
```

Argumen dan Parameter Fungsi

Apa perbedaan Argumen dan Parameter fungsi? Menurut FAQ Programming di dokumentasi Python, adalah sebagai berikut:

> Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what types of arguments a function can accept. For example, given the function definition:

Jadi parameter adalah definisi masukan yang diterima fungsi, dan argumen adalah hal yang Anda masukkan saat memanggil fungsi tersebut.

Contohnya, saat Anda membuat fungsi seperti berikut

```
.   def fungsi_saya (a, b, c):
.       #lakukansesuatudenganabc
```

maka a,b,c adalah parameter. Namun saat Anda memanggilnya

```
.   fungsi_saya (1, b=14, c='Dicoding')
```

maka 1, 14, dan 'Dicoding' adalah argumen.

Menulis Method dan Kelas pada Python

Module Python adalah berkas teks berekstensi .py yang berisikan kode Python. Anda dapat mereferensi berkas .py apa pun sebagai modul. Modul-modul umum yang disediakan oleh Python Standard Library dan mungkin sudah terinstal secara default pada instalasi Python Anda. PIP juga dapat dimanfaatkan untuk menginstal modul atau library berikut dengan dependensi yang dibutuhkannya. Anda pun dapat membuat dan menghasilkan modul Python Anda sendiri.

Menulis Modul

Menuliskan modul pada bahasa Python dapat dimulai dengan menuliskan definisi fungsi, kelas, dan variabel yang dapat digunakan kembali pada program lainnya. Misalkan saja kita membuat berkas hello.py yang akan kita panggil di berkas lain.
hello.py

```
# Define a function
def world():
    print("Hello, World!")
```

Jika hello.py dijalankan, maka program tidak akan menjalankan apapun karena world() hanya berupa definisi fungsi, kita belum memanggilnya. Jika kita biasanya memanggil sebuah fungsi dari berkas yang sama di bagian main, kali ini kita akan membuat berkas lain main.py yang seolah mengimpor hello.py. Pastikan hello.py dan main.py berada dalam satu direktori agar dapat diimpor dan dipanggil.
main.py

```
#impor
import hello

#panggil
hello.world()
```

Saat memanggil sebuah fungsi dari modul yang kita impor, jangan lupa untuk menambahkan nama modulnya diikuti tanda titik, baru fungsi yang akan kita panggil. Dalam hal ini karena kita mengimpor hello.py, maka cukup kita tulis import hello, dan saat memanggilnya dengan hello.world(). Selain itu, kita juga dapat menggunakan from ... import ..., dalam hal ini adalah from hello import world dan memanggil fungsinya langsung yakni world().
Sekarang, saat memanggil main.py, maka akan menghasilkan:

```
Hello, World!
```

Menambahkan variabel

Menambahkan variabel pada modul hello, tambahkan variabel nama, misalnya Dicoding.
hello.py

```
def world():
    print("Hello, World!")

nama = "Dicoding"
```

Berikutnya, kita coba cetak variabel nama.
main.py

```
#impor
import hello

#panggil
hello.world()

#cetak
print(hello.nama)
```

Saat Dijalankan:

```
Hello, World!
Dicoding
```

Menambahkan kelas

Contoh yang lain, mari tambahkan kelas di modul hello. Kita akan membuat kelas Reviewer dengan atribut nama dan kelas, serta fungsi review() yang akan mencetak atribut yang telah didefinisikan.
hello.py

```
def world():
    print("Hello, World!")

nama = "Dicoding"

class Reviewer:
    def __init__(self, nama, kelas):
        self.nama = nama
        self.kelas = kelas

    def review(self):
        print("Reviewer " + self.nama + " bertanggung jawab di kelas " +
self.kelas)
```

Tambahkan kelas pada main.py.
main.py

```
#impor
import hello

#panggil
hello.world()

#cetak
print(hello.nama)

#review
diko = hello.Reviewer("Diko", "Python")
diko.review()
```

Seperti umumnya kelas pada bahasa pemrograman lainnya, Fungsi dan Atributnya dapat diakses setelah kita melakukan instansiasi. Fungsi Review adalah fungsi yang melekat pada kelas Reviewer. Kita juga dapat memanggil diko.Nama atau diko.Kelas sebagai atribut yang melekat di kelas tersebut.
Output:

```
Hello, World!
Dicoding
Reviewer Diko bertanggung jawab di kelas Python
```

Lihat kedua variabel "nama" yang dapat menghasilkan dua nilai berbeda, karena nama yang pertama (hello.nama) melekat pada modul, sementara diko. Nama adalah atribut nama pada kelas Reviewer. Anda harus cukup berhati-hati dalam memastikan variabel seperti pada pembahasan fungsi yang lalu.

Implementasi Kode

Seringkali, modul dimanfaatkan untuk dapat memisahkan antara definisi dan implementasi kode. Namun modul juga dapat berfungsi selayaknya program pada umumnya, yang juga langsung mengeksekusi dalam modul itu sendiri. Contohnya, kita buat hello2.py seperti berikut:
hello2.py

```
# Definisi

def world():
    print("Hello, World!")

# Panggil disini
world()
```

Kemudian bersihkan main.py hingga menyisakan import hello2 saja.
main.py

```
import hello2
```

Saat main_program dijalankan, langsung muncul:

```
Hello, World!
```

Sehingga modul dapat digunakan dengan berbagai metode pemanggilan, bergantung pada definisi, maupun implementasi.

Mengakses Modul dari Folder Lain

Jika Anda bekerja dengan beberapa proyek secara paralel, berikut adalah opsi untuk mengakses modul dari folder lain:

Menambahkan path folder

Opsi ini dipilih umumnya di tahap awal pengembangan, sebagai solusi temporer. Untuk mengetahui path pemanggilan utama, Anda perlu bantuan dari modul sys yang sudah tersedia. Impor modul sys di main program dan gunakan fungsi sys.path.append. Misalnya berkas hello.py kita berada di direktori /home/dicoding/ dan main.py di direktori lainnya. Anda cukup menambahkan path /home/dicoding pada main.py seperti di bawah:

```
import sys
sys.path.append('/home/dicoding')

import hello
...
```

Menambahkan modul pada Python Path

Alternatif ini dapat dipilih saat Anda melakukan pemanggilan modul >1x. Pada intinya pilihan ini akan menambahkan modul yang Anda buat pada Path yang diperiksa oleh Python sebagai modul dan paket-paket bawaan. Anda dapat memanfaatkan sys.path kembali untuk mengetahui posisi Anda saat ini.

```
print(sys.path)
```

Anda mungkin akan menerima output seperti berikut, sangat bergantung dengan jenis environment Anda, tapi pilihlah (atau cobalah satu per satu) jika ada beberapa output.

```
'/home/dicoding/my_env/lib/python3.5/site-packages'
```

Pindahkan hello.py pada direktori di atas. Maka Ia akan dikenali sebagai sebuah modul yang dapat diimpor oleh siapa saja dalam environment tersebut.
Pada main_program.py cukup impor.

```
import hello
```

Class

Class merupakan sintaksis di Python yang menyediakan semua fitur-fitur standar dari Pemrograman Berorientasi Objek atau dalam bahasa Inggris disebut dengan Object Oriented Programming (OOP).
Definisi dari kelas menggunakan sintaksis class seperti hanya definisi fungsi yang menggunakan sintaksis def, kemudian perlu dipanggil (dieksekusi) dahulu sebelum dapat digunakan dan memiliki efek pada program.

```
class NamaKelas:
    pass  # gantikan dengan pernyataan-pernyataan, misal: atribut atau metode
```

Pada pemanggilan sintaksis class tersebut, setelah seluruh pernyataan-pernyataan semuanya selesai diproses (didaftarkan sebagai atribut ataupun metode), maka kelas sudah dibuat dan dapat digunakan.
Sebuah kelas sendiri mendukung dua macam operasi:

1. Mengacu pada atribut.
2. Pembuatan instance atau dalam bahasa Inggris disebut instantiation.

Agar lebih jelas, kita akan membahas menggunakan contoh berikut.

```
class Kalkulator:
    """contoh kelas kalkulator sederhana"""
    i = 12345

    def f(self):
        return 'hello world'
```

Dari pembuatan class Kalkulator di atas, di dalamnya ada definisi atribut i dan definisi fungsi f. Proses mengacu atribut yaitu Kalkulator.i dan Kalkulator.f sesuai definisi akan mengembalikan nilai integer dan fungsi. Pada proses mengacu atribut tersebut juga dapat mengubah nilainya, misalnya dengan memberikan bilangan bulat lain ke Kalkulator.i akan mengubah nilai yang ada saat ini.

```
Kalkulator.i = 1024  # maka nilai atribut i dalam Kalkulator berubah dari 12345 menjadi 1024
```

Objek (object: an instance of a class)

Pembahasan berikutnya adalah instantiation dari sebuah class, menggunakan notasi fungsi yaitu dengan kurung buka-kurung tutup, akan menghasilkan sebuah objek. Kemudian hasil instantiation ini biasanya disimpan dalam sebuah variabel dengan nama yang representatif. Berikut ini adalah contoh membuat instance dari class Kalkulator menghasilkan sebuah objek.

```
k = Kalkulator()  # membuat instance dari kelas jadi objek, kemudian disimpan pada variabel k
```

Sebagai hasil instance sebuah class, suatu objek memiliki atribut dan metode yang didapatkan dari class. Sebuah metode atau dalam bahasa Inggris disebut method, adalah sebuah fungsi khusus yang menjadi "milik" suatu objek.
Untuk memanggil metode f dari objek k, hasil instance dari class Kalkulator di atas sebagai berikut.

```
print(k.f())  # akan mencetak hello world ke layar
```

Kenapa metode adalah sebuah fungsi khusus?
Jika diperhatikan kembali fungsi f dalam definisi class Kalkulator memiliki satu argumen bernama self, sedangkan dalam pemanggilan metode dari objek k di atas tidak menggunakan argumen. Apabila f adalah fungsi biasa pada Python tentu pemanggilan ini akan mengembalikan kesalahan (error). Lebih detail mengenai konvensi ini akan dibahas pada bagian metode dari class.
Pembahasan lebih lanjut mengenai metode dari class ada di bagian selanjutnya.

Class' Constructor

Kembali membahas proses instantiation dari class, sering ditemui kebutuhan mengeset nilai awal atau kondisi awal dari atribut yang dimiliki oleh class tersebut, sehingga untuk kebutuhan ini digunakan sebuah fungsi khusus yang biasa disebut sebagai pembangun atau dalam bahasa Inggris disebut constructor. Di Python, fungsi khusus atau metode sebagai constructor ini bernama __init__ atau biasa diucapkan sebagai "double underscore init". Pada saat dilakukan instantiation dari class, metode __init__ ini secara otomatis akan dipanggil di terlebih dahulu.
Berikut adalah definisi class Kalkulator di atas jika diubah dengan menggunakan constructor.

```
class Kalkulator:
    """contoh kelas kalkulator sederhana"""

    def __init__(self):
        self.i = 12345

    def f(self):
        return 'hello world'
```

Nilai dari atribut i tidak terdefinisi pada awal definisi Kalkulator, setelah dilakukan instantiation maka nilai atribut i akan bernilai 12345. Meskipun bisa mendefinisikan variabel i sebagai atribut dari class Kalkulator, tetapi sebaiknya berhati-hati mengenai variabel yang akan terbagi (shared) untuk semua instance dari class, terutama untuk tipe yang dapat berubah (mutable), misalnya list dan dictionary.
referensi: https://docs.python.org/id/3.8/tutorial/classes.html#class-and-instance-variables

```
class KeranjangBelanja:
```

```
    """contoh tidak baik dilakukan dengan definisi variabel terbagi
"""
    isi = []  # menggunakan list di sini akan terbagi untuk semua in
stance. JANGAN DILAKUKAN
```

Lanjut pembahasan constructor, dengan dilengkapi constructor pun proses instantiation tidak berubah dari sebelumnya.

```
k = Kalkulator()  # membuat instance dari kelas jadi objek, kemudian
 disimpan pada variabel k
```

Lebih lanjut tentang constructor, tentu saja untuk mendukung aplikasi yang lebih dinamis maka constructor dapat memiliki parameter yang bisa dikirimkan saat proses instantiation, bahkan parameternya bisa lebih dari satu jika diperlukan.
Pada contoh berikut ini, constructor memiliki parameter i yang bersifat opsional, apabila dalam proses instantiation tidak dikirimkan parameter, secara otomatis i akan diisi nilai bawaan 12345.

```
class Kalkulator:
    """contoh kelas kalkulator sederhana"""

    def __init__(self, i=12345):
        self.i = i  # i adalah variabel pada constructor, self.i ada
lah variabel dari class

    def f(self):
        return 'hello world'
```

Dengan contoh pemanggilan berikut.

```
k = Kalkulator(i=1024)  # melakukan instantiation sekaligus mengisi
 atribut i jadi 1024
print(k.i)             # mencetak atribut i dari objek k dengan kelu
aran nilai 1024
```

Metode (Method)

Pembahasan lebih detail mengenai metode, selain yang dibahas sebelumnya, kita akan membahas 3 jenis metode:

1. Metode dari objek (object method)
2. Metode dari class (class method)
3. Metode secara static (static method)

Pertama kita membahas metode dari objek, seperti yang sempat dijelaskan secara singkat di atas mengenai metode, atau dalam bahasa Inggris disebut method, secara umum metode adalah sebuah fungsi khusus yang menjadi "milik" suatu objek, yakni hasil instantiation dari class.
Salah satu hal khusus yang dimiliki oleh metode dengan adanya argumen bernama self, Anda tentu bertanya-tanya tentang argumen self pada metode-metode dalam kelas tersebut

sebetulnya apa?
Argumen pertama dari metode-metode dalam class, biasa diberikan nama self sebagai suatu konvensi atau standar penamaan, meskipun Anda bisa juga menggunakan nama lain. Bahkan dalam Python tidak ada arti khusus tentang sintaksis self ini, namun sangat disarankan menggunakan konversi ini agar program Python yang Anda buat akan lebih mudah dimengerti oleh pemrogram lainnya.
Seperti yang Anda sudah perkirakan, untuk sebuah metode, sebetulnya dikirimkan objek (hasil instance dari class) sebagai argumen pertamanya, dalam hal ini bernama self.
Misalnya menggunakan contoh di atas, jika k adalah objek hasil instance dari class Kalkulator, saat melakukan pemanggilan metode f.

```
k.f()
```

ekuivalen dengan

```
Kalkulator.f(k)
```

Argumen self pada metode f akan diisi dengan objek hasil instance dari class Kalkulator.
Sebelum kita membahas yang kedua dan ketiga, yakni metode dari class dan metode secara static, Anda tentu mengingat bahwa sebelumnya sudah belajar fungsi-fungsi bawaan (built-in) dari Python, antara lain: open, sorted, int, str, dan sejumlah lainnya. Terkait metode, ada dua fungsi bawaan yang akan kita bahas, yakni classmethod dan staticmethod.
Catatan:
fungsi decorator adalah sebuah fungsi yang mengembalikan fungsi lain, biasanya digunakan sebagai fungsi transformasi dengan "pembungkus" sintaksis @wrapper.
Referensi: https://docs.python.org/id/3.8/glossary.html#term-decorator.
Classmethod adalah sebuah fungsi yang mengubah metode menjadi metode dari class (class method). Dalam penggunaannya, fungsi ini dijadikan sebagai fungsi decorator @classmethod, kemudian pemanggilannya bisa langsung dari class yang terdefinisi ataupun melalui objek.
Metode dari class (class method) menerima masukan class secara implisit sebagai argumen pertama yang secara konvensi diberikan nama cls.
Berdasar contoh yang sama dengan class sebelumnya, berikut adalah metode dari class.

```
class Kalkulator:
    """contoh kelas kalkulator sederhana"""

    def f(self):
        return 'hello world'

    @classmethod
    def tambah_angka(cls, angka1, angka2):
        return '{} + {} = {}'.format(angka1, angka2, angka1 + angka
2)
```

Nampak pada kode, sesuai konvensi ada metode yang menggunakan argumen pertama self, sedangkan untuk class method menggunakan konvensi argumen pertama cls.
Untuk melakukan pemanggilan dari class, dilakukan seperti berikut, dimana argumen pertama cls sudah mendapatkan masukan class Kalkulator.

```
. Kalkulator.tambah_angka(1, 2)  # tanpa perlu memberikan masukan unt
uk argumen cls
```

Metode dari class (class method) juga dapat dipanggil dari objek, hasil instantiation dari class Kalkulator, contohnya mirip seperti pemanggilan metode dari objek (object method).

```
. k = Kalkulator()
. print(k.tambah_angka(1, 2))
```

Staticmethod adalah sebuah fungsi yang mengubah metode menjadi metode statis (static method). Dalam penggunaannya, fungsi ini dijadikan sebagai fungsi decorator @staticmethod, kemudian pemanggilannya bisa langsung dari class yang terdefinisi ataupun melalui objek. Metode statis (static method) tidak menerima masukan argumen pertama secara implisit. Untuk Anda yang pernah memrogram Java atau C++, metode statis ini mirip seperti yang ada di bahasa pemrograman tersebut.
Berdasar contoh yang sama dengan class sebelumnya, berikut adalah metode statis.

```
. class Kalkulator:
.     """contoh kelas kalkulator sederhana"""
.
.     def f(self):
.         return 'hello world'
.
.     @staticmethod
.     def kali_angka(angka1, angka2):
.         return '{} x {} = {}'.format(angka1, angka2, angka1 * angka
2)
```

Nampak pada kode, tidak ada argumen pertama yang implisit seperti halnya pada dua metode sebelumnya.
Pemanggilan dari class seperti halnya pemanggilan fungsi biasa.

```
. a = Kalkulator.kali_angka(2, 3)
. print(a)
```

Metode statis (static method) juga dapat dipanggil dari objek, hasil instantiation dari class Kalkulator, mirip seperti pemanggilan fungsi biasa meskipun dipanggil dari objek.

```
. k = Kalkulator()
. a = k.kali_angka(2, 3)
. print(a)
```