Konsep Pemrograman

Modul Praktikum

Yurizal Susanto

2018

Contents

Lab Komputer dan Lab Informatika IF/FTI-ITI

Tata Tertib Praktikum Komputer

1. Penilaian Praktikum ini dilakukan terhadap 3 hal yaitu:
    1. Penilaian hasil dari praktikum
    2. Kehadiran
    3. Sikap dan tingkah laku saat praktikum
2. Dalam praktikum ini peserta harus memperhatikan hal-hal sebagai berikut:
    1. Kehadiran peserta praktikum harus **100%**, semua modul praktikum harus dilaksanakan. Apabila peserta tidak hadir pada jadwal yang ditentukan, peserta harus dilaksanakan pada jam yang lain dengan konsekuensi membayar biaya perawatan sebesar Rp. 20.000,- (dua puluh ribu rupiah) per modul. Peserta yang tidak melaksanakan salah satu modul dan tidak mengganti pada jam yang lain, maka nilai praktikum modul tersebut adalah **0**.
    2. Bagi yang memerlukan praktikum tambahan, dapat menggunakan fasilitas laboratorium (bila ada yang kosong) dengan membayar biaya perawatan sebesar Rp. 1.500,- (seribu lima ratus rupiah) per jam atau bagian dari 1 jam.
    3. Keterlambatan lebih dari 15 menit tidak diperkenankan mengikuti praktikum.
    4. Setiap praktikum, Kartu Praktikum harus dibawa dan diserahakn pada asisten yang bertugas.
    5. Duduklah pada tempat yang telah ditentukan sesuai dengan nomor yang tertera di Kartu Praktikum.
    6. Tas, buku, *flashdisk*, dll; harus diletakkan pada tempat yang telah disediakan *kecuali Buku Petunjuk Praktikum.*
    7. Peserta harus berpakaian rapih dan tidak diperkenankan memakai sandal.
    8. Ruangan praktikum merupakan ruangan ber-AC. Di mana tidak seorang pun diperkenankan untuk merokok, membawa makanan, dan minuman.

9. Peralatan komputer yang ada adalah peralatan yang berharga. Kecerobohan peserta yang menyebabkan kerusakan alat harus ditanggung oleh peserta itu sendiri.
10. Praktikan dilarang mengganti atau mengubah perangkat lunak (*software*) atau kata sandi yang sudah ada.
11. Selama praktikum berlangsung, jaga kesopanan dan ketenangan supaya tidak mengurangi manfaat dari praktikum anda. Sangsi atas pelanggaran ini dapat mempengaruhi nilai anda.
12. Setelah praktikum selesai, harap bersihkan meja praktikum dari sampah-sampah dan membuang sampah tersebut pada tempat yang telah disediakan.
13. Sebelum meninggalkan ruangan, komputer dan monitor yang digunakan harus dalam keadaan mati.
14. Peserta harus meninggalkan ruangan bila ada aba-aba untuk selesainya waktu praktikum.
15. Nilai praktikum merupakan komponen penentu nilai akhir.

Tata Tertib Penggunaan Petunjuk Praktikum

Buku Petunjuk Praktikum adalah *milik Institut*, tidak diberikan tetapi *dipinjamkan* selama satu semester. Apabila rusak/hilang maka peserta dikenakan denda sebesar Rp. 10.000,- (sepuluh ribu rupiah) atau mengganti buku tersebut.

Prakata

Cara Penggunaan Modul

Untuk setiap baris tulisan yang menggunakan huruf *monospace*, anda harus mengetik perintah tersebut. Jika awal baris dimulai dengan tanda `$`.

```
$ phyton halo.py
Halo, Dunia!
```

Artinya perintah itu dimasukkan ke Terminal/*Command Line* dan baris selanjutnya adalah hasil dari perintah tersebut.

Jika awal baris dimulai dengan `»>`, gunakan Python *interpreter* untuk memasukkan perintah.

```
>>> 4*10 + 2
42
```

Jika ada kumpulan baris dengan huruf *monospace* seperti dibawah ini, dimana baris pertama dimulai dengan `# Program nama_program.py`, masukkan perintah tersebut ke sebuah file bernama `nama_program.py`.

```
1 # Program halo.py
2 print('Halo, Dunia!')
```

1 Jalan Program

Apa itu Program?

Program adalah rentetan instruksi yang diberikan kepada komputer untuk memberi tahu cara melakukan komputasi. Instuksinya beragam, mulai dari perhitungan matematika, mengganti atau mencari text, mengolah gambar, atau memainkan video. Secara garis besar instruksi terdiri dari kategori yang sama namun implementasi dari setiap bahasa pemrograman berbeda, contoh kategori intruksi:

input Ambil data dari *keyboard*, *file*, jaringan, atau perangkat lain.

output Tampilkan data ke layar, simpan ke *file*, kirim melalui jaringan, dll.

math Melakukan operasi matematika sederhana.

conditional execution Cek untuk kondisi dan menjalankan kode yang sesuai dengan kondisi.

repetition Melakukan instruksi berulang kali, terkadang dengan variasi.

Menjalankan Python

Di praktikum ini kita menggunakan bahasa pemrograman Python. Python adalah bahasa pemrograman tingkat tinggi yang multiguna. Pertama kali muncul tahun 1991, oleh Guido van Rossum. Kini, Python dikembangkan oleh *Python Software Foundation*.

Untuk menjalankan Python, buka aplikasi untuk mengakses *Command Line*. Pada Windows anda dapat membuka Windows Powershell. Pada Linux dan MacOS, gunakan aplikasi terminal emulator. MacOS, gunakan iTerminal; dan Linux bisa membuka Gnome Terminal, KDE Terminal, dan lain-lain.

```
$ python
```

Ketik perintah di atas lalu tekan `enter`, dan akan muncul *prompt* seperti di bawah ini.

```
Python 3.6.5 (default, May  3 2018, 20:09:11)
[GCC 7.3.0] on linux
Type ”help”, ”copyright”, ”credits” or ”license” for more information.
>>>
```

Tiga baris pertama akan menunjukkan informasi tentang sistem operasi yang digunakan. Selain itu, praktikan harus memeriksa versi dari Python, di contoh versi yang digunakan adalah `3.6.5`. Jika versi Python yang muncul bukan versi 3, tekan `Ctrl+Z` lalu `Enter` (di Windows) atau `Ctrl+D` (di Linux dan MacOSX untuk keluar. Lalu gunakan perintah `python3` untuk mencoba masuk ke Python versi 3.

Baris terakhir dari *prompt* menunjukan bahwa *interpreter* siap dalam menerima perintah.

Program Pertama

Secara tradisional, program pertama yang ditulis saat belajar bahasa pemrograman baru adalah "Halo, Dunia!". Di Python kalian dapat mengetik perintah seperti ini:

```
>>> print('Halo, Dunia!')
```

Lalu tekan `Enter` untuk melihat hasil perintah, dalam kasus ini hasilnya adalah:

```
Halo, Dunia!
```

Di atas adalah contoh penggunaan fungsi `print`. Tanda kurung menandakan bahwa `print` adalah sebuah fungsi; tanda kutip diantara tulisan "Halo, Dunia!" menandakan awal dan akhir kata yang akan ditampilkan, tanda kutip tidak akan muncul pada tampilan.

Operator Aritmatika

Materi selanjutnya adalah Operator Aritmatika. Python memberikan semua operator aritmatika standar seperti pertambahan, pengurangan, perkalian, dan pembagian.

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
>>> 84 / 2
42.0
```

Berbeda dengan operator lain, hasil operator pembagian membuat nilai yang dihasilkan menjadi nilai desimal.

Selain itu ada juga operator khusus untuk perpangkatan dan modulo, yaitu operator `**` dan `%`:

```
>>> 6**2 + 6
42
```

```
>>> 13%7 * 7
42
```

Nilai dan Tipe

Nilai adalah unsur datar yang dikelola oleh program, seperti huruf atau angka. Beberapa nilai yang sudah kita lihat adalah `42`, `42.0`, dan `Halo, Dunia!`.

Nilai tersebut memiliki tipe masing-masing: `42` adalah *integer*, `42.0` adalah angka desimal (atau *float*), dan `Halo, Dunia!` adalah *string*.

Jika tidak yakin tipe dari sebuah nilai, *interpreter* bisa memberi tahu kita:

```
>>> type(42)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Halo, Dunia!')
<class 'str'>
```

Bagaiman untuk nilai seperti `'42'` dan `'42.0'`? Mereka seperti angka, tetapi dikelilingi oleh tanda kutip seperti *string*.

```
>>> type('42')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Variabel

Salah satu fitur paling penting dari bahasa pemrograman adalah Variabel. Variabel adalah nama yang mengacu ke sebuah nilai.

Untuk membuat variabel, hanya perlu menggunakan operator penugasan `=`.

```
>>> pesan = 'Ini adalah pesan rahasia'
>>> x = 10
>>> pi = 3.14
```

Nama variabel biasanya dipilih agar memiliki makna dari nilai yang dikandungnya, hal ini juga menjadi dokumentasi sederhana agar kita tahu apa yang program kita lakukan. Ada beberapa ketentuan nama variabel:

1. Nama variabel hanya boleh menggunakan karakter a–z, A–Z, 0–9, dan garis bawah (_).

2. Tidak boleh diawali dengan karakter angka; contoh `3serangkai`.
3. Tidak boleh sama dengan *keywords* yang digunakan bahasa Python.

Table 1.1: Daftar *keywords* Python

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	
assert	else	import	pass	
break	except	in	raise	

Gunakan variabel dengan hanya menulis namanya, dan *interpreter* akan secara otomatis menggantinya dengan nilai yang kita simpan di dalam variabel tersebut.

```
>>> print('Anda mendapat pesan:', pesan)
Anda mendapat pesan: Ini adalah pesan rahasia
>>> x * 2
20
```

Latihan

1. Gunakan interpreter Python sebagai kalkulator. Jika sebuah segitiga siku-siku memiliki alas 5.34 cm dan tinggi 10.5 cm, hitunglah luas segitiga tersebut.
2. Gunakan interpreter Python sebagai kalkulator. Seorang konsumen membeli makanan dengan jumlah total pembayaran Rp. 231.500 tanpa pajak. Di restoran tersebut setiap pembayaran makanan dikenakan pajak sebanyak 7.5%. Berapa total biaya yang harus dibayar konsumen?
3. Buat *essay* pendek tentang sejarah pembuatan bahasa pemrograman Python dan kegunaan bahasa pemrograman Python (berikan contoh produk atau usaha yang menggunakan Python dan bagaimana mereka menggunakannya). *Essay* ditulis tangan pada kertas Folio dengan panjang minimal 1,5 halaman.

2 Fungsi dan Mode Skrip

Fungsi

Dalam pemrograman fungsi adalah rentetan intrusksi yang memiliki nama. Saat kita mendefinisikan fungsi, pertama kita memberi nama, lalu kita memberikan rentetan instruksinya. Sehingga nanti kita bisa memanggil rentetan fungsi tersebut dengan nama yang diberikan.

Kita sudah melihat beberapa contoh pemanggilan fungsi.

```
>>> help(len)
```

Nama fungsi yang dipanggil di atas adalah `help`. Pernyataan dalam tanda kurung disebut sebagai *argumen* dari fungsi. `help` sendiri adalah fungsi untuk menampilkan deskripsi dari fungsi yang menjadi argumennya.

Sangat sering *programmer* berkata fungsi "mengambil" argumen dan "mengembalikan" hasil. Hasil dari fungsi biasa kita sebut *nilai balik*.

Mode Skrip

Pada praktikum sebelumnya kita menggunakan *interpreter* untuk memasukkan perintah yang ingin dijalankan, ini disebut mode interaktif. Namun, penggunaan *interpreter* membuat perintah yang ditulis tidak dapat digunakan kembali, tanpa harus mengetik ulang perintah tersebut. Python mendukung mode eksekusi di mana semua perintah yang ingin dijalankan, dimasukkan ke dalam sebuah file berekstensi `.py`.

Ketik program di bawah ini dengan *plaintext editor* (Notepad++, Atom, Sublime), dengan nama `halo.py`.

```
1 # Program halo.py
2 print('Halo Dunia!')
3 nama = input('Masukkan nama: ')
4 print('Nama saya', nama)
```

Untuk menjalankannya buka PowerShell atau Terminal anda dan jalankan perintah dibawah ini:

```
$ python halo.py
# atau
$ python3 halo.py
```

Selanjutnya program akan berjalan, saat bagian `Masukkan nama:`, program akan berhenti dan menunggu masukkan *keyboard* yang diakhiri dengan tombol `Enter`. Setelah tombol `Enter` ditekan, nilai masukkan *keyboard* sebelumnya akan masuk ke variabel `nama`. Lalu di baris ke 3, isi variabel `nama` ditampilkan.

Contoh eksekusi program:

```
Halo Dunia!
Masukkan nama: John Doe
Nama saya John Doe
```

Perbedaan antara mode skrip dan mode interaktif adalah bagaimana program akan terus berjalan dalam mode interaktif sampai baris akhir perintah tereksekusi. Sehingga jika kita ingin menggunakan variabel yang nilainya kita isi, kita harus menggunakan fungsi `input` yang sudah dicontohkan.

Fungsi `input` menghasilkan nilai bertipe `string` sehingga untuk menghasilkan nilai lain, perlu bantuan fungsi `int`, dan `float` untuk mengubah `string` ke masing-masing tipe.

Fungsi Pustaka

Setiap bahasa pemrograman memiliki kumpulan fungsi yang dipaketkan bersama bahasa pemrograman tersebut. Kumpulan fungsi itu biasa disebut dengan pustaka (*library*). Dalam Phyton banyak jenis pustaka yang bisa kita gunakan. Mulai dari pustaka matematika, kriptografi, tanggal dan waktu, dan lain-lain.

```
1 # Program hitung_lingkaran.py
2 import math
3
4 radius = float(input('Masukkan radius lingkaran: '))
5 luas = math.pi * radius**2
6 keliling = 2 * math.pi * radius
7
8 print('Luas lingkaran:', luas, end=' ')
9 print('Keliling Lingkaran:', keliling)
```

Melihat contoh di atas, untuk menggunakan pustaka kita dapat menggunakan kata `import`. Setelah itu baru nama pustaka yang ingin digunakan, contohnya `math`. Pustaka `math`, sesuai namanya, berisi beragam fungsi matematika beberapa diantaranya `sin`, `cos`, `tan`, `sqrt`, `log`, dan lain-lain.

Selain menggunakan `import` untuk menggunakan pustaka, kita juga bisa mengunakan syntax `from ... import ...` untuk penggunaan pustaka jika kita ingin kemampuan memilih fungsi apa saja yang ingin dipakai.

```
# Program hitung_lingkaran_alt.py
from math import sin, cos, tan

derajat = float(input('Derajat: '))

hasil_sin = sin(derajat)
hasil_cos = cos(derajat)
hasil_tan = tan(derajat)

print('sin({}): {:.3}'.format(derajat, hasil_sin))
prnt('cos({}): {:.3}'.format(derajat, hasil_cos))
print('tan({}): {:.3}'.format(derajat, hasil_tan))
```

Dengan menggunakan syntax `from ... import ...`, kita bisa memilih fungsi atau nilai dari pustaka yang ingin kita pakai. Sehingga saat dipanggil, kita tidak perlu menulis nama pustaka.

Membuat Fungsi

Selain fungsi awal bawaan dari Python, seorang programmer juga biasanya membuat fungsi sendiri.

Pembuatan fungsi diawali dengan kata `def`, dilanjutkan dengan nama fungsi (ketentuan nama fungsi sama dengan nama variabel), lalu dilanjutkan dengan karakter buka, lalu parameter jika perlu, dan tutup kurung, diakhiri dengan titik dua `:`.

```
def spam():
    print('Spam!Spam!Spam!')

def spam_canggih(n):
    print('Spam!' * n)

def spam_tercanggih(n, kata='Spam!'):
    print(kata * n)

spam()
spam_canggih(10)
spam_tercanggih(5)
spam_tercanggih(5, 'Merdeka!')
```

Seperti yang diberitahukan sebelumnya, fungsi dapat menerima argumen. Sebagai contoh misalnya kita ingin menampilkan kata `Spam!` lebih dari 3 kali, maka deklarasi fungsi `spam` jadi seperti fungsi `spam_canggih`.

Lalu parameter fungsi bisa diberikan nilai bawaan, jadi jika parameter pemanggilan kurang, yang terjadi bukan *error*. Fungsi akan otomatis menggunakan nilai bawaan dari parameter.

Latihan

1. Buat fungsi untuk menentukan jarak suatu kota berdasarkan skala dan jarak pada peta yang dimasukkan. Buat juga fungsi kebalikannya; yaitu menentukan jarak kota di peta jika diberikan masukkan jarak asli dan skalanya.

   Contoh penggunaan:

   ```
   jarak_asli(2, 100000)
   jarak_peta(10.75, 100000)
   ```

   Rumus:

   jarak asli = jarak peta $\div$ skala

   jarak peta = jarak asli $\cdot$ skala

   Simpan program pada file, dan berikan contoh penggunaan fungsi tersebut dengan fungsi `input` dan `print`.

2. Buat fungsi dari pernyataan matematika berikut ini: $f(x) = 2x^2 * 4x + 3$

3 Kondisional

Operator Relasi dan Logika

Untuk melakukan instruksi kondisional, program biasanya menggunakan ekspresi *boolean*, yaitu ekspresi yang memiliki nilai `True` atau `False`. Contoh; di bawah ini kita membandingkan nilai dengan operator `==`, yang membandingkan 2 nilai dan menghasilkan nilai `True` jika kedua nilai sama, atau `False` untuk sebaliknya:

```
>>> 7 == 7
True
>>> 3 == 7
False
>>> type(7 == 7)
<class 'bool'>
>>>
```

Hasil dari operator `==` adalah `bool` yang merupakan salah satu tipe data dalam bahasa pemrograman Python untuk nilai `True` atau `False`.

Operator `==` merupakan salah satu dari opearator relasi. Operator relasi lainnya yang ada di Python adalah:

```
x != y          # x tidak sama dengan y
x > y           # x lebih besar dari y
x < y           # x lebih kecil dari y
x >= y          # x lebih besar sama dengan y
x <= y          # x lebih kecil sama dengan y
```

Penggunaan operator relasi sama seperti yang ada pada matematika, hanya saja simbol yang digunakan berbeda.

Terkadang kondisi pada pernyataan kondisional itu kompleks dan tidak cukup hanya menggunakan operator relasi saja. Terkadang kita ingin `x` itu lebih kecil dari sesuatu, tetapi tidak sama dengan nilai tertentu. Kebutuhan seperti itu tidak dapat dipenuhi hanya dengan operator relasi. Kita harus merangkai kondisinya dengan operator logika.

Operator logika ada 3; `and`, `or`, dan `not`. Contohnya; `x > 0 and x < 10` itu benar jika nilai x lebih besar dari 0 dan lebih kecil dari 10.

Lalu `n%2 == 0 or n%3 == 0` itu benar jika `n` dapat dibagi habis oleh angka 2 atau 3.

Terakhir, `not` berfungsi untuk menegasi ekspresi *boolean*. `not (x == y)` itu benar jika `x == y` hasilnya `False`, yang artinya `x` tidak sama dengan `y`.

Pernyataan Kondisional

Untuk menulis program yang berguna, program harus punya kemampuan untuk mengecek kondisi dan mengubah intruksi yang dijalankan sesuai kondisi yang ada. Pernyataan kondisional memberikan kita kemampuan ini. Contoh sederhana:

```
n = int(input("Angka: "))

if n % 2 == 0:
    print(n, "adalah bilangan genap.")
else:
    print(n, "adalah bilangan ganjil.")
```

Terkadang kita juga butuh cabang eksekusi lebih dari 2, untuk itu kita butuh melakukan yang namanya rantai kondisi.

```
# Program menu.py

def halo_dunia():
    print('Halo, Dunia!')

def teriak():
    kalimat = input('Masukkan kalimat: ')

    print(kalimat.upper())

def perbandingan():
    x = int(input('Masukkan angka X: '))
    y = int(input('Masukkan angka Y: '))

    if x < y:
        print('X lebih kecil dari Y')
    elif x > y:
        print('X lebih besar dari Y')
    else:
        print('X sama dengan Y')

print('a. Program Halo Dunia')
print('b. Program Meneriakan Kalimat')
print('c. Program Perbandingan')
```

```
25
26 pilihan = input('Masukkan pilihan [a/b/c] atau sembarang untuk keluar: ')
27
28 if pilihan == 'a':
29     halo_dunia()
30 elif pilihan == 'b':
31     teriak()
32 elif pilihan == 'c':
33     perbandingan()
34 else:
35     print('Pilihan tidak ada...')
36
37 print('Program selesai.')
```

Dicontohkan oleh program di atas, untuk melakukan rantai kondisi kita menggunakan *syntax* `elif` yang merupakan kependekan dari *else if*. Setiap kondisi dicek sesuai urutan, jika kondisi pertama salah, kondisi yang selanjutnya akan dicek, dan seterusnya. Tidak ada batasan jumlah penggunaan `elif`, hanya saja `elif` hanya bisa digunakan setelah ada kondisi `if`.

Fungsi Bernilai

Selama ini fungsi yang kita buat adalah fungsi *void*, atau fungsi tanpa nilai. Fungsi bernilai adalah fungsi yang mengembalikan hasil prosesnya. Contoh dari fungsi bernilai yang sudah dipakai adalah `int`, `str`, dan `float`.

```
>>> a = int('42')
>>>
```

`int` menerima argumen dan mengeluarkan nilai dengan tipe data *integer* yang mendekati argumen.

```
1  # Program fungsi_luas_lingkaran.py
2  from math import pi
3
4  def luas_lingkaran(r):
5      luas = pi * r**2
6      return luas
7
8  def bukan_negatif(n):
9      if n > 0:
10         return True
11     else:
12         return False
```

```
13
14 radius = float(input('Masukkan radius lingkaran: '))
15 luas = 0
16
17 if bukan_negatif(radius):
18     luas = luas_lingkaran(radius)
19     print('Luas lingkaran: {:.3}'.format(luas))
20 else:
21     print('Radius tidak boleh negatif')
```

Fungsi bernilai dibuat dengan *syntax* `return`. Saat program menemukan `return`, fungsi akan berhenti dan mengeluarkan nilai.

Refactor Kode

Kegiatan *Refactoring* adalah kegiatan menyederhanakan program tanpa mengubah tujuan dari program tersebut. *Refactoring* berguna untuk mendapatkan program yang mudah dibaca dan diurus.

Sebagai contoh fungsi `luas_lingkaran` di atas akan kita *refactor*

```
>>> from math import pi
>>> def luas_lingkaran(r):
...     return pi * r**2
...
>>> luas_lingkaran(10)
314.1592653589793
>>>
```

Fungsi `luas_lingkaran` di-*refactor* dengan menghapus variabel sementara yang memagang hasil luas lingkaran.

Refactoring juga bisa dilakukan ke fungsi `bukan_negatif`.

```
>>> def bukan_negatif(n):
...     if n > 0:
...         return True
...     return False
...
>>> bukan_negatif(-3)
False
>>>
```

Pernyataan `else` dapat kita hapus, tanpa mengubah tujuan dari fungsi `bukan_negatif`. Selain *refactor* dengan menghapus pernyataan `else`, kita dapat menghapus pernyataan

`if` juga. Karena tujuan dari fungsi `bukan_negatif` adalah mengecek apakah `n > 0`, kita bisa langsung mengembalikan nilai `n > 0`.

```
>>> def bukan_negatif(n):
...     return n > 0
...
>>> bukan_negatif(-1)
False
>>> bukan_negatif(42)
True
>>>
```

Latihan

1. Buatlah fungsi untuk menentukan huruf tersebut huruf vokal atau bukan. Jika benar fungsi tersebut mengeluarkan nilai True atau jika salah akan mengeluarkan nilai false.
2. Buatlah fungsi untuk menentukan huruf tersebut huruf konsonan atau bukan. Jika benar fungsi tersebut mengeluarkan nilai True atau jika salah akan mengeluarkan nilai false. Nilai plus bagi yang intruksi dalam fungsinya hanya satu baris.

   Contoh penggunaan:

   ```
   >>> huruf_vokal('a')
   True
   >>> huruf_vokal('z')
   False
   >>> huruf_konsonan('a')
   False
   >>> huruf_konsonan('z')
   True
   ```

   Simpan 2 fungsi di atas dalam satu file program, dan berikan contoh penggunaannya.

4 Pengulangan

Pengulangan Kondisional

Biasanya program memiliki kemampuan untuk melakukan pengulangan intruksi. Pengulangan intruksi ini biasanya memiliki kondisi sehingga tidak terjadi pengulangan tanpa henti. Dalam bahasa pemrograman Python kita menggunakan *syntax* `while` untuk melakukan pengulangan dengan kondisi.

```
# Program tebak_angka.py
from random import randint, seed

seed()
min = 0
max = 100

print('Selamat Datang di Permainan Tebak Angka')
x = int(input('Masukkan angka: '))
jawaban = randint(min, max)

while x != jawaban:
    if x > jawaban:
        print('Lebih kecil...')
    elif x < jawaban:
        print('Lebih besar...')
    x = int(input('Masukkan angka: '))

print('Selamat! Anda berhasil menebak angka', jawaban)
```

Program di atas adalah program permainan sederhana untuk menebak angka. Angka yang ditebak digenerasi secara acak oleh komputer, dan kita harus memasukkan angka yang sama dengan jawaban. Cara kerja `while` hampir sama dengan cara kerja `if`, jika ekspresi setelah `while` bernilai `True` atau bukan nol, maka intruksi setelahnya akan berjalan terus menerus sampai `while` mendapatkan ekpresi bernilai `False`.

Selain dengan ekspresi bernilai `False`, kita juga bisa memaksa pengulangan untuk berhenti dengan *syntax* `break`.

```
max = int(input('Maksimal pengulangan: '))
```

```
2
3  x = 0
4  while True:
5      x = x + 1
6      if max == 3:
7          continue
8      if max == 5:
9          break
10     print(x)
```

`while True` akan menghasilkan pengulangan tanpa henti. Untuk menghentikannya kita harus menambahkan `break`. `continue` digunakan untuk melanjutkan pengulangan tanpa menghiraukan perintah dibawahnya.

Pengulangan Sekuensial

Di Python, ada sebuah pengulangan spesial yaitu pengulangan sekuensial. Pengulangan sekuensial adalah pengulangan terhadap nilai sekuensial, *syntax* yang digunakan adalah `for`. Nilai sekuensial sendiri salah satunya yang pernah kita temui adalah `string`. Karena `string` adalah kumpulan nilai karakter sekuensial.

Selain menggunakan nilai sekuensial, kita juga bisa menggunakan fungsi `range` yang menggenerasi nilai sekuensial untuk kita dalam bentuk urutan angka.

```
1  # Program cari_karakter.py
2
3  def cari(kata, huruf):
4      for index in range(len(kata)):
5          if kata[index] == huruf:
6              return index
7      return -1
8
9  kata = input('Masukkan kata: ')
10 huruf = input('Masukkan huruf yang dicari: ')
11
12 index = cari(kata, huruf)
13 if index == -1:
14     print(huruf, 'tidak ditemukan pada', kata)
15 else:
16     print(huruf, 'ada pada index', index)
```

Fungsi `range` menghasilkan nilai sekuensial dari 0 sampai nilai sebanyak `x` argumen yang dimasukkan. Misal `range(5)` akan menghasilkan nilai `[0, 1, 2, 3, 4]`.

Fungsi Rekursif

Untuk mengerti rekursi, anda harus mengerti rekursi

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Karena itu fungsi ini terkadang digunakan sebagai pengganti pengulangan. Cara pembuatannya sama seperti fungsi biasa, tetapi didalam intruksi fungsi kita juga memanggil nama fungsi yang dibuat.

```
>>> def spam():
...     print('Spam!')
...     spam()
...
>>> spam()
Spam!
Spam!
Spam!
.
.
.
Spam!
Spam!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in spam
  File "<stdin>", line 3, in spam
  File "<stdin>", line 3, in spam
  [Previous line repeated 992 more times]
  File "<stdin>", line 2, in spam
RecursionError: maximum recursion depth exceeded while calling a Python object
Spam!
>>>
```

Sebagai programmer, kita harus hati-hati dalam penggunaan fungsi rekursif. Seperti dicontohkan di atas, fungsi rekursif kita berjalan tanpa henti. Hingga akhirnya *interpreter* Python memanggil *error* bahwa pemanggilan fungsi kita mencapai batas kedalaman rekursi.

Contoh fungsi rekursif dalam matematika adalah faktorial.

$0! = 1$

$n! = n(n-1)!$

$0! = 1$ biasanya disebut `base case` dalam fungsi rekursif. Guna dari `base case` adalah sebagai pembatas dari fungsi rekursif. Jika dituliskan dalam Python menjadi seperti dibawah ini:

```
>>> def faktorial(n):
...     if n == 0:
...         return 1
```

Merujuk pada definisi faktorial di atas, lalu kita tulis bagian $n! = n(n-1)!$:

```
>>> def faktorial(n):
...     if n == 0:
...         return 1
...     else:
...         return n * faktorial(n-1)
...
>>> faktorial(0)
1
>>> faktorial(4)
24
>>>
```

Jika dinyatakan dalam kalimat, fungsi faktorial bisa dituliskan seperti di bawah ini, sebagai contoh `faktorial(3)`:

`n` sama dengan 3, hasilnya faktorial 3 `*` faktorial(n-1), faktorial(n-1) dijalankan
    `n` sama dengan 2, hasilnya faktorial 2 `*` faktorial(n-1), faktorial(n-1) dijalankan
        `n` sama dengan 1, hasilnya faktorial 1 `*` faktorial(n-1), faktorial(n-1) dijalankan
            `n` sama dengan 0, hasilnya 1
        `n` sama dengan 1, hasilnya 1 `*` 1
    `n` sama dengan 2, hasilnya 2 `*` 1
`n` sama dengan 3, hasilnya 3 `*` 2

Penggunaan fungsi faktorial terkadang bisa digantikan dengan melakukan pengulangan biasa. Meskipun ada beberapa kasus khusus yang hanya bisa menggunakan faktorial, misalnya fungsi *Ackermann*.

Latihan

1. Ubah program `tebak_angka.py` di atas, menjadi kebalikan program tersebut. Pengguna memasukkan angka yang harus ditebak, lalu komputer akan mencoba menebak angka tersebut. Jika salah, kalian harus memberitahu komputer apakah nilai yang harus ditebak itu lebih tinggi atau lebih kurang dari tebakkan mereka. Nilai plus bagi mereka yang dapat memberitahukan berapa banyak tebakkan yang diperlukan komputer untuk menebak angka yang benar.
2. Buat program yang menerima masukkan angka, lalu menambahkan angka yang dimasukkan. Program akan berhenti menerima angka jika mendapatkan angka

0. Hasil dari program adalah menampilkan berapa angka yang masuk, total pertambahan angka tersebut, dan rata-ratanya.

5 *List*

Pembuatan

List adalah tipe data yang dapat menyimpan banyak nilai. *List* juga merupakan sebuah tipe data rangkaian, seperti *String*.

```
>>> warna = [”merah”, ”hitam”, ”kuning”, ”biru”]
>>> warna[0]
'merah'
>>> warna[1:3]
['hitam', 'kuning']
>>> a = [1, 2, 3]
>>> b = [True, False, a]
>>> kosong = []
>>> print(a, b, kosong)
[1, 2, 3] [True, False, [1, 2, 3]] []
>>>
```

Mengakses *list*

Gunakan `for` untuk mengakses *list* secara keseluruhan satu per satu.

```
1  # Program cetak_daftar.py
2
3  def cetak_daftar(daftar):
4      for nama in daftar:
5          print(nama.title())
6
7  daftar_nama = ['anto ferdinanto', 'budi pambudi', 'yanti sriyanti']
8  cetak_daftar(daftar_nama)
```

Manipulasi List

```python
# Program anggota_ukm.py

def tambah_anggota(nama, daftar):
    daftar.append(nama)

def hapus_anggota(nama, daftar):
    index = daftar.index(nama)
    del daftar[index]

def cek_status(nama, daftar):
    if nama in daftar:
        print(nama, 'terdaftar sebagai anggota')
    else:
        print(nama, 'bukan bagian UKM')

def cetak_anggota(daftar):
    for nama in daftar:
        print(nama.title())

selesai = False
daftar_anggota = []

while not selesai:
    print('Program pencatatan anggota UKM')
    print('1. Cetak daftar anggota')
    print('2. Tambah anggota')
    print('3. Hapus anggota')
    print('4. Cek status anggota')
    print('5. Keluar')

    pilihan = input('Pilihan menu: ')

    if pilihan == '1':
        cetak_anggota(daftar_anggota)
    elif pilihan == '2':
        nama = input('Nama anggota baru: ')
        tambah_anggota(nama, daftar_anggota)
    elif pilihan == '3':
        nama = input('Nama anggota: ')
        hapus_anggota(nama, daftar_anggota)
    elif pilihan == '4':
```

```
42          nama = input('Nama anggota: ')
43          cek_status(nama, daftar_anggota)
44      else:
45          selesai = True
```

Gunakan metode `append` untuk menambahkan elemen ke dalam *list*. Untuk menghapus elemen di *list* gunakan syntax `del`, seperti yang dicontohkan program di atas.

List dan *String*

List dan *string* sangat berdekatan fungsinya, kita dapat membuat *list* dari *string*, begitu juga sebaliknya.

```
>>> musik = 'jazz, rock, pop, ballad, hiphop, waltz'
>>> daftar_musik = musik.split(', ')
>>> daftar_musik
['jazz', 'rock', 'pop', 'ballad', 'hiphop', 'waltz']
>>> huruf = 'abcdefghijkl'
>>> daftar_huruf = list(huruf)
>>> daftar_huruf
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']
>>> daftar_kucing = ['bobtail', 'shorttail', 'ragdoll', 'persian', 'balinese']
>>> kucing = '-'.join(daftar_kucing)
>>> kucing
'bobtail-shorttail-ragdoll-persian-balinese'
>>>
```

Contoh program yang menggunakan *list* dan *string*:

```
1  # Program generator_password.py
2  from secrets import choice
3
4  import string
5
6  def generator_password(n):
7      karakter = list(string.ascii_letters + string.digits)
8      password = []
9
10     for i in range(n):
11         x = choice(karakter)
12     password.append(x)
13
14     return ''.join(password)
15
```

```
16 print('Program Pembuat Password')
17 n = int(input('Jumlah karakter password:')
18 print('Password anda:', generator_password(n))
```

Tugas

1. Ubah program `generator_password.py` di atas untuk membatasi minimal panjang *password* menjadi 8 (kembalikan nilai *string* kosong jika panjang *password* tidak memenuhi batas). Berikan juga fitur untuk membangun *password* dengan karakter tanda baca seperti `-@?!$()/><` (Bantuan: Gunakan konstanta `string.punctuation`).

   Nilai plus bagi yang bisa mengimplementasikan generator *password* dengan memastikan ada minimal 1 huruf kapital di dalam *password* yang dihasilkan, dan tidak ada karakter yang mengulang.

6 *Dictionary*

Pemetaan *Dictionary*

Sebuah *dictionary* itu hampir sama dengan *list*; namun lebih umum. Jika *index* dari sebuah *list* harus `integer`, tidak dengan *dictionary*.

Dalam *dictionary*, *index* disebut sebagai *key*. Di mana *key* itu dipetakan ke sebuah nilai. Nilai dapat berupa jenis tipe data apapun.

```
>>> eng2indo = dict()
>>> eng2indo['morning'] = 'pagi'
>>> eng2indo['night'] = 'malam'
>>> print(eng2indo)
{'morning': 'pagi', 'night': 'malam'}
```

Manipulasi *Dictionary*

Manipulasi *dictionary* memeliki beberapa *syntax* yang sama dengan manipulasi *list*. *Dictionary* dapat dihapus, diubah, dan ditambahkan.

```
>>> nilai = {'ipa': [80, 70], 'matematika': [70, 75], 'bahasa': [80, 85]}
>>> 'bahasa' in nilai
True
>>> 'ips' in nilai
False
>>> del nilai['bahasa']
>>> len(nilai)
2
>>>
```

Akses *dictionary* dengan *for*

Dictionary juga bisa diakses dengan `for`. `for` akan melakukan pengulangan pada setiap *key* yang ada pada *dictionary*.

```python
# Program buku_telepon.py

def cetak_nomor(daftar):
    for nama in daftar:
        print(nama, daftar[nama])

def cari_nomor(nama, daftar):
    if nama in daftar:
        return daftar[nama]
    return None

def tambah_nomor(nama, nomor, daftar):
    daftar[nama] = nomor

buku_telepon = dict()

print('Program Buku Telepon')

while True:
    nama = input('Masukkan nama: ')

    if nama == '':
        cetak_nomor(buku_telepon)
        continue

    nomor = cari_nomor(nama, buku_telepon)
    if not nomor:
        print('Nomor tidak ditemukan untuk', nama)
        nomor_baru = input('Masukkan nomor telepon baru: ')
        print('Memasukkan nomor baru...')
        tambah_nomor(nama, nomor_baru, buku_telepon)
        continue

    print('Nomor telepon', nama, 'adalah:', nomor)
```

Copy by Reference

Copy by Reference adalah konsep menyalin nilai pada variabel dengan petunjuk tempat nilai tersebut disimpan. Konsep ini terjadi jika kita ingin menyalin nilai variabel yang menyimpan tipe data *mutable* seperti *list* dan *dictionary*.

```
>>> tempat = ['hogwart', 'midgard', 'prontera', 'konoha']
```

```
>>> tempat_baru = tempat
>>> tempat_baru.append('death star')
>>> print(tempat_baru)
['hogwart', 'midgard', 'prontera', 'konoha', 'death star']
>>> print(tempat)
['hogwart', 'midgard', 'prontera', 'konoha', 'death star']
>>>
```

Untuk melakukan *copy by value*, duplikasi nilai, gunakan *deep copy*.

```
>>> tokoh = ['linus torvalds', 'dennis ritchie', 'ken thompson']
>>> tokoh_baru = tokoh[:]
>>> tokoh_baru = tokoh.copy() # Alternative dari intruksi di atas
>>> tokoh_baru.append('guido van rossum')
>>> print(tokoh_baru)
['linus torvalds', 'dennis ritchie', 'ken thompson', 'guido van rossum']
>>> print(tokoh)
['linus torvalds', 'dennis ritchie', 'ken thompson']
>>>
```

Latihan

1. Ubah program `buku_telepon.py` di atas dan tambahkan fitur baru untuk menghapus, mengubah nama atau nomor, dan memperbaiki tampilan saat pencetakan nomor telepon seperti di bawah ini:

   ```
   saya              082131231356
   bapak             089341312358
   ibu               081295049536
   tukang ayam bakar 085945345607
   langganan ojek    084313230596
   ```

2. Buat fungsi `terjemahkan` yang dapat menerima *string* dan menerima *dictionary*. Fungsi tersebut dapat menerjemahkan *string* yang diterima berdasarkan isi dari *dictionary*.

   ```
   >>> kata = 'i love you'
   >>> kamus = {'love': 'cinta', 'i': 'saya'}
   >>> kata_baru = terjemahkan(kata, kamus)
   >>> print(kata_baru)
   'saya cinta you'
   ```

   Simpan pada sebuah file, dan berikan contoh penggunaannya.

7 *Tuple*

Menggunakan *Tuple*

Tuple adalah tipe data yang dapat digunakan untuk menyimpan banyak nilai seperti *list*. Namun, *tuple* adalah tipe data yang *immutable*.

```
>>> angka = 1, 2, 3
>>> angka = (1, 2, 3)
>>> type(angka)
<class 'tuple'>
>>> print(angka[2])
3
>>> angka[0] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Sehingga *tuple* tidak berguna untuk menyimpan data yang banyak dan ingin dimanipulasi.

Fungsi dengan Banyak Nilai Balik

Sebuah fungsi biasanya menghasilkan nilai balik. Fungsi yang kita buat biasanya memiliki 1 nilai balik. Python mendukung fungsi untuk memiliki banyak nilai balik, hal ini karena kita dapat mengembalikan *tuple* sebagai nilai balik.

```
>>> a = divmod(7, 2)
>>> print(a)
(3, 1)
>>> a, b = divmod(7, 2)
>>> print(a, b)
3 1
```

Fungsi `divmod`, fungsi membagi dan modulo, menghasilkan 2 nilai, kita dapat menyimpan nilai kembali dengan 1 variabel atau dengan 2 variabel.

Contoh program yang menghasilkan nilai balik lebih dari 1.

```python
# Program penghitung_nilai.py

def input_nilai(n):
    nilai = []

    for i in range(n):
        x = float(input('Masukkan nilai: '))
        nilai.append(x)

    return nilai

def average(s):
    return sum(s) / len(s)

def min_dan_max(s):
    return min(s), max(s)

def sum_dan_avg(s):
    return sum(s), average(s)

print('Program menghitung nilai')
n = int(input('Berapa banyak nilai yang ingin dimasukkan? '))
nilai = input_nilai(n)

minimal, maksimal = min_dan_max(nilai)
total, rata = sum_dan_avg(nilai)

print('Nilai terendah:', minimal, 'Nilai tertinggi:', maksimal)
print('Total nilai:', total, 'Nilai rata-rata:', rata)
```

Matrix

Dalam Python tidak ada tipe dasar Matrix, kita dapat menggunakan pustaka khusus, atau untuk data yang sedikit kita bisa menggunakan *list* dalam *list*.

```python
# Program operasi_matrix.py

def input_matrix(panjang, lebar):
    matrix = []

    for x in range(panjang):
        matrix.append([])
```

```
        for y in range(lebar):
            deskripsi = 'Masukkan angka matrix baris {} kolom {}: '.format(x+1, y+1)
            angka = int(input(deskripsi))
            matrix[x].append(angka)

    return matrix

def pertambahan_matrix(mat_a, mat_b):
    if len(mat_a) != len(mat_b) or len(mat_a[0]) != len(mat_b[0]):
        raise RuntimeError('Ordo matrix tidak sama')

    hasil = []
    for baris in range(len(mat_a)):
        hasil.append([])
        for kolom in range(len(mat_a[0])):
            hasil[baris].append(mat_a[baris][kolom] + mat_b[baris][kolom])

    return hasil

def print_matrix(matrix):
    for baris in matrix:
        for kolom in baris:
            print('{:4}'.format(kolom), end='')
        print()

print('Program pertambahan matrix')
h = int(input('Masukkan banyak baris: '))
w = int(input('Masukkan banyak kolom: '))

mat_a = input_matrix(h, w)
print('Matrix A:')
print_matrix(mat_a)

mat_b = input_matrix(h, w)
print('Matrix B:')
print_matrix(mat_b)

hasil = pertambahan_matrix(mat_a, mat_b)
if hasil != None:
    print('Matrix hasil:')
    print_matrix(hasil)
```

Latihan

1. Lengkapi program `operasi_matrix.py` dengan menambahkan fungsi untuk melakukan perkalian dan pengurangan.

8 File

Membaca File

Program yang kita pakai sering kali digunakan untuk memanipulasi file dokumen yang ada. Sekarang kita akan belajar bagiamana cara menerapkannya di program buatan sendiri.

Contoh program yang membaca file:

```
1  # Program pembaca_file.py
2
3  def cetak_file(path):
4      # Buka file
5      f = open(path, 'r')
6
7      text = f.read()
8      print(text)
9
10     # Tutup file
11     f.close()
12
13 def cetak_file_perbaris(path):
14     f = open(path, 'r')
15
16     for baris in f:
17         print(baris)
18
19     f.close()
20
21 print('Program Pembaca File')
22 nama = input('Nama file: ')
23
24 cetak_file(nama)
25 cetak_file_perbaris(nama)
```

Untuk menulis sesuatu ke sebuah file, kita harus mengubah mode yang digunakan oleh fungsi `open` dalam membuka file menjadi `write/w` atau `append/a`.

Table 8.1: Daftar mode pembacaan Python

Mode	Keterangan
`r`	Mode membaca (*default*)
`w`	Mode menulis
`a`	Mode menambah
`b`	Mode biner
`t`	Mode teks (*default*)
`+`	Mode pembaruan

Menulis File

Contoh program yang menulis file:

```
1 # Program menulis_file.py
2 nama = input('Nama file: ')
3
4 with open(nama, 'w') as f:
5     teks = input()
6     f.write(teks + '\n')
```

Program di atas akan meminta masukkan 2 kali. Yang pertama, program akan meminta nama file. Yang kedua, program akan meminta teks yang akan dimasukan ke dalam file.

Jalur File

Dalam manipulasi file dalam pemrograman ada yang namanya istilah jalur relatif dan absolut. Relatif artinya pencarian file dimulai dari direktori sekarang kita berada, sedangkan absolut dimulai dari awal direktori pada file sistem.

```
Nama file relatif:
file.txt


Nama file absolut:
C:\some_folder\file.txt
```

Lalu ada lagi masalah tentang perbedaan pembatas pada nama file antara Windows dan banyak sistem operasi:

```
Nama file di windows:
C:\some_folder\file.txt
```

```
Nama file di sistem operasi lain:
/some_folder/file.txt
```

Untuk masalah di atas bisa menggunakan `Path` dari pustaka `pathlib` untuk mengatasinya:

```
>>> from pathlib import Path
>>> folder = Path('Documents')
>>> nama_file = folder / 'folder_baru' / 'file.txt'
>>> print(nama_file)
```

Perintah di atas akan membuat jalur file yang sesuai dengan sistem operasi yang sedang digunakan.

Latihan

1. Ubah program `buku_telepon.py` di praktikum 7 untuk dapat menyimpan nomor telepon ke dalam file.
2. Buat sebuah program yang dapat menyimpan nama yang dimasukan dengan format `nomor. nama yang di input`. Jika program ditutup dan dibuka kembali, harus mengulang dari nomor terakhir yang dimasukan. Misal nomor terakhir adalah `11. Suyono`, program harus memulai dari nomor `12. nama yang di input`. Gunakan metode `readline()` untuk membaca satu per satu.

9 Pustaka

Membuat Pustaka

Pustaka pada Python biasanya adalah kumpulan file pada folder tertentu yang memberikan fungsi atau nilai yang dapat digunakan. Pustaka biasa disebut *package* atau *module* dalam Python.

Package dibuat dengan mengelompokkan *module* Python dalam satu direktori. Membuat *package* pada Python itu mudah. Buat sebuah direktori dan tambahkan file `__init__.py` didalamnya.

Buat susunan direktori seperti di bawah ini:

```
convert/
└── convert
    ├── convert.py
    └── __init__.py
```

Kita akan buat pustaka yang memiliki fungsi untuk mengkonversi jumlah uang dari satu mata uang ke mata uang yang lain dan fungsi untuk mendapat nilai konversi per 1 dalam mata uang. Kita akan gunakan API yang dipublikasikan oleh Bank Sentral Eropa untuk mendapat nilai konversi. https://exchangeratesapi.io/

```
GET https://api.exchangeratesapi.io/latest HTTP/1.1
```

```
{
  "base": "EUR",
  "date": "2018-04-08",
  "rates": {
    "CAD": 1.565,
    "GBP": 0.87295,
    "USD": 1.2234,
    ...
  }
  }
```

Di atas adalah contoh data yang akan kita dapatkan dari API. Data tersebut nantinya akan diubah menjadi *dictionary*.

Isi `convert.py` dengan kode di bawah ini.

```
# Requests adalah pustaka untuk mengambil data dari Internet.
import requests

def get_rate(from_currency, to):
    '''get_rate adalah fungsi untuk mengembalikan nilai tukar per 1 mata uang.
    from_currency adalah mata uang yang ingin dikonversikan.
    to adalah mata uang tujuan konversi.
    Parameter from_currency dan to hanya menerima kode mata uang.
    Contohnya: USD, IDR, JPY, GBP
    '''
    url = 'https://api.exchangeratesapi.io/latest'
    # Parameter url
    data = {'base': from_currency, 'symbols': to}
    # Unduh data dari API dengan parameter
    r = requests.get(url, params=data)
    # Ubah data jadi dictionary dan ambil bagian yang diperlukan, lalu
    # mengembalikannya
    return r.json()['rates'][to]

def convert(from_currency, to, amount):
    '''convert adalah fungsi yang mengembalikan amount di kalikan nilai konversi
    mata uang.
    from_currency adalah mata uang yang ingin dikonversikan.
    to adalah mata uang tujuan konversi.
    Parameter from_currency dan to hanya menerima kode mata uang.
    Contohnya: USD, IDR, JPY, GBP
    Parameter amount adalah jumlah mata uang yang ingin dikonversikan ke to.
    '''
    rate = get_rate(from_currency, to)
    return amount * rate
```

Ganti isi `__init__.py` menjadi seperti dibawah ini.

```
1 from . import get_rate, convert
```

Setelah itu tambahkan file baru bernama `setup.py`, seperti di susunan direktori di bawah ini:

```
convert/
├── convert
│   ├── convert.py
│   └── __init__.py
└── setup.py
```

Isi dari `setup.py` adalah cara bagaimana Python untuk memasang pustaka kita ke

komputer.

```
from setuptools import setup, find_packages

setup(name='convert', # Nama pustaka
      version='0.1', # Versi pustaka
      description='Konversi mata uang', # Deskripsi singkat package
      author='Aslab ITI', # Nama pembuat package
      author_email='aslab@iti.ac.id',
      license='MIT', # Lisensi dari pustaka
      packages=find_packages(), # Pustaka apa saja yang dipasang
      install_requires = ['requests'], # Pustaka lain yang diperlukan
      zip_safe=False)
```

Lalu untuk memasang pustaka kita, kita dapat menggunakan perintah `pip` dari terminal atau *commandline*.

```
$ cd convert
$ pip install --user .
```

Perintah `pip` akan mencari file `setup.py` untuk mulai memasang pustaka kita.

Setelah selesai dipasang kita dapat mencoba memanggilnya dengan interpreter:

```
>>> import convert
>>> convert.get_rate('JPY', 'IDR')
132.1042663472
>>> convert.convert('IDR', 'USD', 75000)
4.9850
```

10 Data Science dengan Python

Numpy (Numerical Python)

Numpy adalah sebuah pustaka yang memberikan kemampuan untuk melakukan pembuatan *array* multidimensi dan juga koleksi fungsi yang dapat memanipulasi *array* tersebut. Numpy biasa digunakan dengan menggabungkannya dengan *package* lain seperti SciPy (Scientific Python) dan Mat-Plotlib, kombinasi ini sering digunakan sebagai pengganti MatLab.

Array Numpy

```
>>> import numpy as np
>>> a = np.array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
```

Untuk membuat *array* dalam Numpy perlu menggunakan *list* yang sudah pernah diajarkan sebelumnya. Perlu diingat bahwa kita mengubah *list* menjadi Numpy *array*.

```
>>> a = np.array(1, 2, 3) # Salah
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: only 2 non-keyword arguments accepted
>>> a = np.array([1, 2, 3]) # benar
>>> a
array([1, 2, 3])
```

Selain *array* 1 dimensi, bisa juga dibuat array 2 dimensi dengan memanfaatkan *nested list*.

```
>>> b = np.array([ [1,2,3], [4,5,6] ])
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
```

Untuk membuat *array* multidimensi, Numpy memberikan beberapa fungsi bantuan seperti `ones()` dan `zeros()`. Yang masing-masing membuat *array* multidimensi dengan isi setiap indeksnya 1 atau 0.

```
>>> m = np.zeros([3, 4])
>>> m
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> n = np.ones([2, 6])
>>> n
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

Operator Numpy

Objek *array* yang dibuat Numpy juga bisa digunakan dalam operasi matematika.

```
>>> a = np.array([1, 2, 3])
>>> b = a + 3
>>> b
array([4, 5, 6])
>>> b
>>> b= a * 2
array([2, 4, 6])
```

Operasi matematika antara *array* Numpy dan skalar (integer) menghasilkan operasi yang dilakukan terhadap setiap elemen. Jika kita tidak menggunakan Numpy, untuk melakukan operasi pada setiap elemen kita perlu menggunakan *loop.

Contoh operasi matematika setiap elemen tanpa Numpy.

```
>>> b = []
>>> for x in a:
...     b.append(x**2)
...
>>> b
[2, 4, 6]
```

Operasi matematika juga bisa dilakukan antara 2 objek *array*. Nanti hasilnya akan sama seperti operasi matematika pada matriks.

```
>>> x = np.array([[0, 1], [2, 3]])
>>> y = np.array([[2, 2], [2, 2]])
>>> x + y
array([[2, 3],
```

```
       [4, 5]])
>>> x * y
array([[0, 2],
       [4, 6]])
```

Berbeda dari operasi pada umumnya, operator `*` mengkalikan pada setiap elemen. Jika ingin menghasilkan hasil perkalian matriks, gunakan operator `@`.

```
>>> x @ y
array([[ 2,  2],
       [10, 10]])
```

Akses indeks array

Elemen pada *array* Numpy bisa diakses dengan indeksnya, sama seperti *list*. *Slicing* juga bisa dilakukan pada *array*.

```
>>> a = np.arange(10) ** 3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[3]
27
>>> a[3:7]
array([ 27,  64, 125, 216])
```

Indeks *array* multidimensi bisa diakses dengan menambahkan angka dan koma.

```
>>> c = np.array([
... [1, 2, 3, 4, 5, 6],
... [6, 5, 4, 3, 2, 1]])
>>> c
array([[1, 2, 3, 4, 5, 6],
       [6, 5, 4, 3, 2, 1]])
>>> c[0]
array([1, 2, 3, 4, 5, 6])
>>> c[1, 4]
2
```

Pandas

Data science biasanya berurusan dengan data. Data tersebut bisa dalam bentuk series dan tabular. Python memiliki pustaka yang bisa digunakan untuk memanipulasi data dari sebuah file bernama Pandas. Pandas dapat membaca isi file CSV (comma separated value) atau spreadsheet (xlsx atau ods).

Membaca File

Buat sebuah berkas dengan nama `gaji.csv`, dan masukkan isi berikut:

```
Nama,Kelamin,Gaji
Yogi,L,3500000
Amel,P,4000000
Levi,P,3600000
Fitri,P,4200000
Agi,L,3700000
Linus,L,5000000
Rick,L,3900000
Dea,P,4700000
Lowe,L,4100000
Udin,L,2500000
```

Lalu buka *interpreter* Python untuk membaca berkas yang baru dibuat tersebut.

```
>>> import pandas as pd
>>> df = pd.read_csv('gaji.csv')
```

`read_csv()` digunakan untuk membaca file dan mengembalikan nilai dengan tipe DataFrame. Selain membaca CSV, bisa juga pandas digunakan untuk membaca spreadsheet, Excel dan Libreoffice Calc, dengan fungsi `read_excel()`.

Menampilkan DataFrame

Pandas menyiapkan beberapa fungsi dan metode untuk menampilkan nilai yang ada dalam DataFrame.

```
>>> df
    Nama Kelamin     Gaji
0   Yogi       L  3500000
1   Amel       P  4000000
2   Levi       P  3600000
3  Fitri       P  4200000
4    Agi       L  3700000
5  Linus       L  5000000
6   Rick       L  3900000
7    Dea       P  4700000
8   Lowe       L  4100000
9   Udin       L  2500000
>>> df.head(3)
   Nama Kelamin     Gaji
0  Yogi       L  3500000
```

```
1  Amel       P  4000000
2  Levi       P  3600000
>>> df.tail(3)
   Nama Kelamin     Gaji
7   Dea       P  4700000
8  Lowe       L  4100000
9  Udin       L  2500000
>>> df.Gaji
0    3500000
1    4000000
2    3600000
3    4200000
4    3700000
5    5000000
6    3900000
7    4700000
8    4100000
9    2500000
Name: Gaji, dtype: int64
```

Metode `head()` digunakan untuk menampilkan n-baris pertama dari DataFrame. Sedangkan, `tail()` digunakan untuk menampilkan n-baris terakhir dari DataFrame. Selain itu, nama kolom juga bisa dipanggil untuk menampilkan isi dari kolom itu saja.

Manipulasi DataFrame

Isi dari DataFrame bisa diurutkan berdasarkan kolom:

```
>>> df.sort_values(by='Gaji')
    Nama Kelamin     Gaji
9   Udin       L  2500000
0   Yogi       L  3500000
2   Levi       P  3600000
4    Agi       L  3700000
6   Rick       L  3900000
1   Amel       P  4000000
8   Lowe       L  4100000
3  Fitri       P  4200000
7    Dea       P  4700000
5  Linus       L  5000000
```

Dalam menampilkan isi DataFrame, bisa diseleksi kolom apa yang ditampilkan.

```
>>> df.loc[:, ['Nama', 'Gaji']]
    Nama     Gaji
```

```
0   Yogi  3500000
1   Amel  4000000
2   Levi  3600000
3  Fitri  4200000
4    Agi  3700000
5  Linus  5000000
6   Rick  3900000
7    Dea  4700000
8   Lowe  4100000
9   Udin  2500000
```

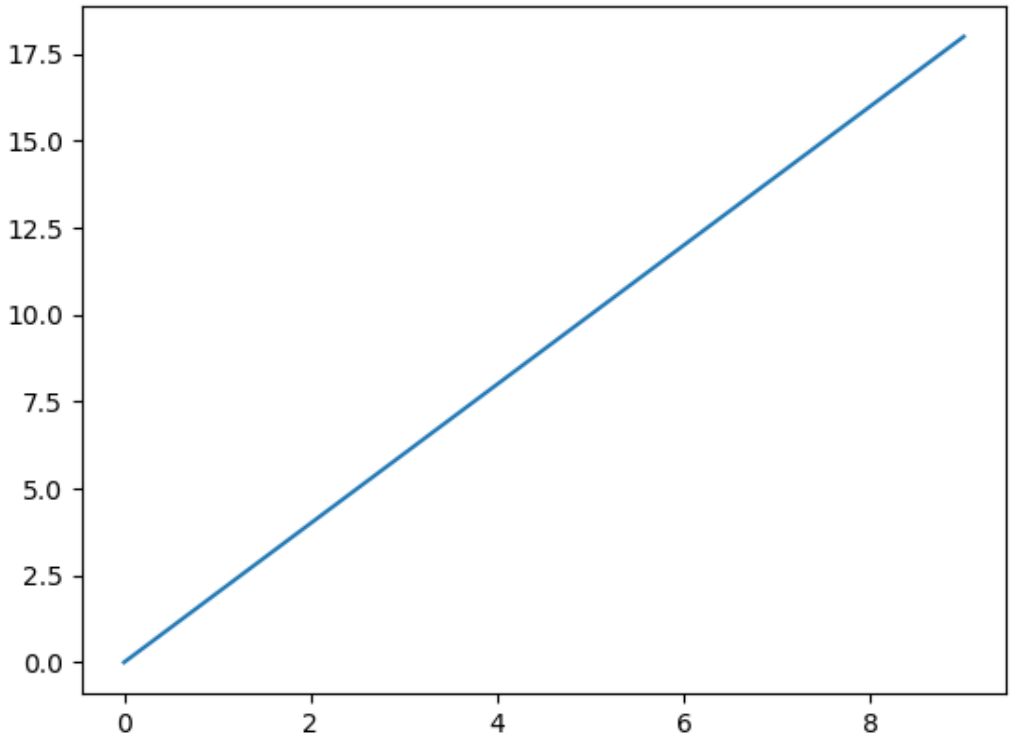
Selain itu, bisa juga untuk menampilkan DataFrame yang sesuai dengan nilai Boolean.

```
>>> df[df.Kelamin == 'L']
    Nama Kelamin     Gaji
0   Yogi       L  3500000
4    Agi       L  3700000
5  Linus       L  5000000
6   Rick       L  3900000
8   Lowe       L  4100000
9   Udin       L  2500000
```

Matplotlib

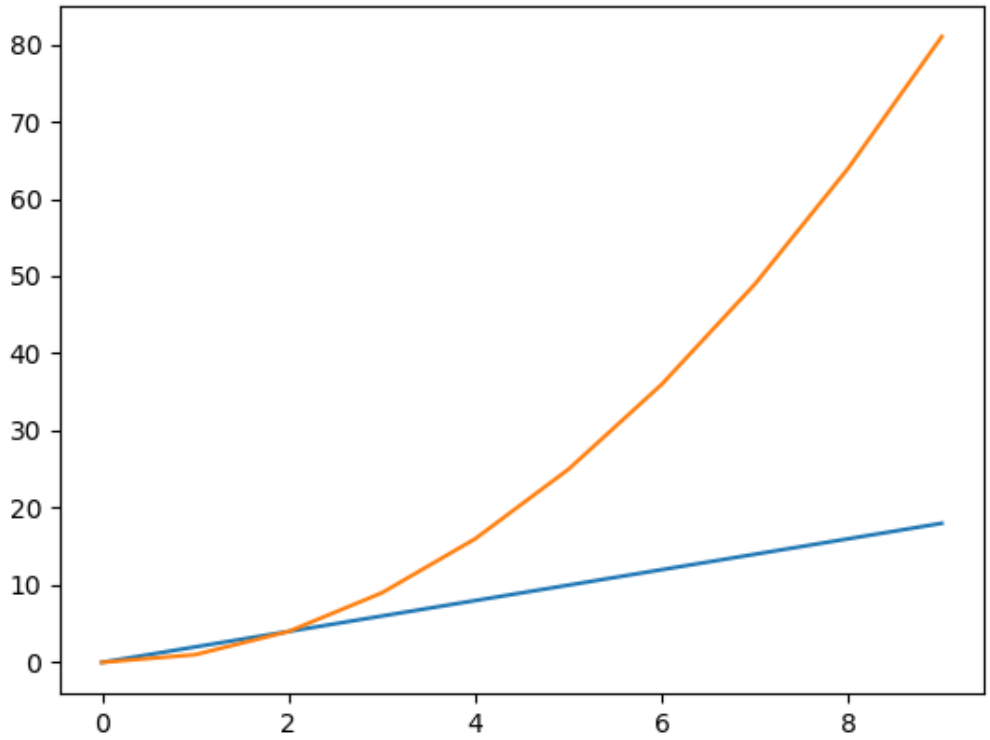
Matplotlib adalah pustaka Python yang digunakan untuk melakukan *plotting* data dalam gambar 2 dimensi.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(10)
>>> y = 2 * x
>>> plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x7f6f2adf2400>]
>>> plt.show()
```



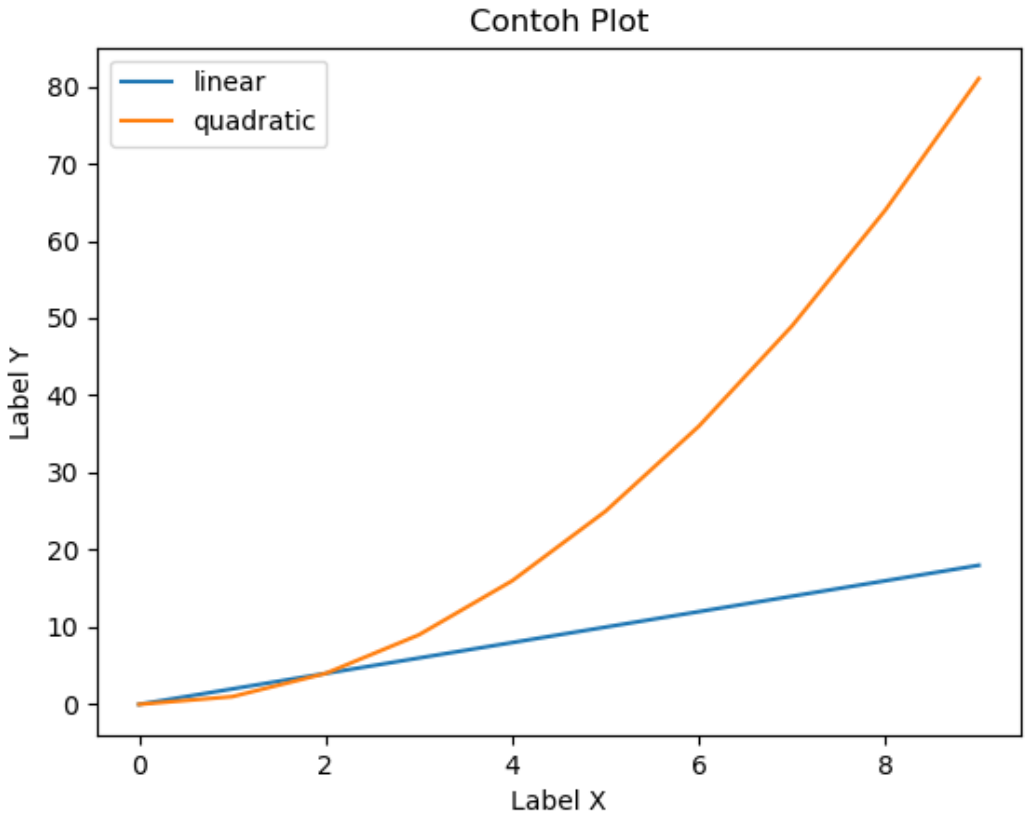
Kode di atas akan menampilkan visualisasi angka linear. Bagaimana jika kita ingin menampilkan beberapa garis?

```
>>> x = np.arange(10)
>>> y = 2 * x
>>> z = x ** 2
>>> plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x7f6f2b11c710>]
>>> plt.plot(x, z)
[<matplotlib.lines.Line2D object at 0x7f6f2b1912b0>]
>>> plt.show()
```



Hasil gambar visualisasi bisa ditambahkan keterangan untuk membantu dalam mengetahui makna garis yang di-*plot*-kan.

```
>>> x = np.arange(10)
>>> y = 2 * x
>>> z = x ** 2
>>> plt.plot(x, y, label="linear")
[<matplotlib.lines.Line2D object at 0x7f6f2b084080>]
>>> plt.plot(x, z, label="quadratic")
[<matplotlib.lines.Line2D object at 0x7f6f2b8d03c8>]
>>> plt.xlabel("Label X")
Text(0.5,0,'Label X')
>>> plt.ylabel("Label Y")
Text(0,0.5,'Label Y')
>>> plt.title("Contoh Plot")
Text(0.5,1,'Contoh Plot')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f6f2b0ea908>
>>> plt.show()
```



Pandas dan Matplotlib

DataFrame yang dihasilkan oleh Pandas dapat dengan mudah di-*plot*-kan dengan matplotlib.

```
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> df = pd.read_csv('gaji.csv')
>>> df.plot.bar(x='Nama')
<matplotlib.axes._subplots.AxesSubplot object at 0x7fda66861320>
>>> plt.show()
```

DataFrame bisa divisualisasikan dengan bantuan metode `plot()`. Plot bawaan dari metode `plot()` adalah plot garis. Untuk membuat visualisasi yang lain, bisa menggunakan metode di bawah ini:

- `plot.bar()`
- `plot.hist()`
- `plot.pie()`
- `plot.box()`

- `plot.area()`
- `plot.scatter()`