# Modul Praktikum
# Pemrograman Python

Ali Akbar Septiandri
Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Al Azhar Indonesia

2017

# Praktikum Pemrograman Python

## Deskripsi

Praktikum Pemrograman Python merupakan bagian dari mata kuliah Pemrograman Python. Modul-modul dalam praktikum ini akan dikerjakan selama 1 SKS, yang diambil dari 3 SKS mata kuliah Pemrograman Python. Teknis pelaksanaan praktikum ini adalah sejalan dengan kuliah Pemrograman Python, yaitu setelah penjelasan setiap bab dari perkuliahan Pemrograman Python.

## Tujuan Umum

Praktikum ini diharapkan dapat membantu mahasiswa untuk mengerti penggunaan berbagai pustaka dan kakas dalam bahasa Python untuk menghasilkan suatu aplikasi. Beberapa materi yang diberikan dalam perkuliahan Pemrograman Python sangat berkaitan dengan kuliah lainnya, misalnya **Pemrograman Berorientasi Objek** atau **Web Dinamis**. Oleh karena itu, materi praktikum untuk kuliah ini hanya difokuskan pada implementasinya pada Python dengan sedikit penjelasan secara umum.

## Peraturan

Semua modul praktikum dikerjakan secara individual. Secara umum, Anda tidak diperkenankan menggunakan pustaka selain yang disebutkan secara eksplisit pada tiap modul praktikum. Hanya modul bawaan Python saja yang diperbolehkan untuk digunakan secara bebas dalam praktikum ini.

## Petunjuk Pengerjaan Tugas Akhir

Dalam beberapa tugas akan diberikan masukan berupa file kasus uji. Keluaran dari program tersebut beserta kode program yang digunakan harus dikompresi menggunakan format zip dan diunggah melalui situs `http://elearning2.uai.ac.id/`.

# Daftar Isi

# Modul 1

# Hello, World!

## 1.1 Tujuan

Modul ini bertujuan untuk pengenalan awal dengan bahasa Python. Beberapa poin yang akan dibahas antara lain:

1. Pengenalan *syntax* dasar dalam bahasa Python
2. Melakukan *control flow & looping*
3. Pengenalan struktur data standar dalam Python
4. Penulisan fungsi dalam Python

## 1.2 Teori

Pemrograman dalam bahasa Python tidak menggunakan kurung kurawal "{}" sebagai penanda blok kode. Selain itu, *syntax* Python juga tidak memerlukan tanda titik koma ";" di akhir baris. Penanda blok kode digantikan dengan indentasi, yang lebih disarankan untuk diisi dengan spasi alih-alih tab. Berdasarkan standar PEP-8, disarankan untuk menggunakan **4 spasi** untuk menandakan blok kode untuk keseragaman.

Python menggunakan prinsip *dynamically typing*, artinya tidak perlu ada deklarasi tipe untuk tiap variabel atau keluaran dari fungsi yang dibuat. Meski begitu, Python tetap mengenal konsep *casting*. String dalam Python dapat dilihat juga sebagai *array of characters*.

Dalam Python, *array* lebih dikenal dengan nama *list*. *List* dapat difungsikan sebagai *stack* (LIFO) maupun *queue* (FIFO). Indeks dalam Python selalu dimulai dari nol, sedangkan pengaksesan elemen terakhir dari *list* dapat menggunakan indeks negatif. *Slicing* juga dapat dilakukan dengan menggunakan indeksnya saja.

Pembuatan fungsi dalam Python tidak memerlukan deklarasi tipe keluaran, seperti yang telah disebutkan di atas, demikian pula halnya dengan argumen dari fungsi. Python juga memungkinkan fleksibilitas pembuatan fungsi yang lebih tinggi dengan memperbolehkan argumen opsional dan mendefinisikan jumlah argumen taktentu. Fungsi kosong pada Python dibuat dengan menggunakan kata kunci `pass`.

## 1.3 Praktikum

Listing 1.1: latihan1.py

```
#!/usr/bin/env python

print "Nama:",
nama = raw_input()

print "Umur:",
umur = raw_input()

print "{} berumur {} tahun".format(nama, umur)
```

Listing 1.2: latihan2.py

```
#!/usr/bin/env python
import sys

nama = sys.argv[1]
umur = sys.argv[2]
print "{} berumur {} tahun".format(nama, umur)
```

Listing 1.3: segitiga.py

```
#!/usr/bin/env python
import sys

def luasSegitiga(alas, tinggi):
    return alas * tinggi / 2

_, alas, tinggi = sys.argv
print "Segitiga dengan alas {} cm dan tinggi {} cm".format(alas, tinggi
    )
print "Luas: {} cm2".format(luasSegitiga(float(alas), float(tinggi)))
```

Listing 1.4: bmi.py

```
#!/usr/bin/env python

print "Weight:", # 185
weight = raw_input()
print "Height:", # 81.6
height = raw_input()

bmi = float(weight) / ((float(height)/100) ** 2)
if bmi < 18.5:
    print "underweight"
elif 18.5 <= bmi < 25:
    print "normal"
else:
    print "overweight"
```

Listing 1.5: chars.py

```
#!/usr/bin/env python

print "Masukkan nama Anda:",
nama = raw_input()

a, e, i = 0, 0, 0
for c in nama:
    if c == 'a':
        a += 1
    elif c == 'e':
        e += 1
    elif c == 'i':
        i += 1

print "Terdapat %d huruf 'a'" % a
print "Terdapat %d huruf 'e'" % e
print "Terdapat %d huruf 'i'" % i
```

## 1.4 Tugas Akhir

### 1.4.1 Deskripsi

Buatlah sebuah program untuk menghitung luas $n$ bangun datar yang terdiri dari:

1. Persegi
2. Persegi panjang
3. Segitiga
4. Lingkaran (gunakan 3.14 untuk nilai pi)
5. Trapesium

Permintaan masukan harus disesuaikan dengan bangun datar yang akan dihitung luasnya, e.g. jika persegi, maka program hanya akan meminta satu masukan, i.e. sisi. Setelah didapatkan $n$ luas bangun datar, urutkan luas bangun datar dari terkecil hingga terbesar dan hitung total luas bangun datar tersebut!

### 1.4.2 Contoh

Listing 1.6: luas.in

```
4
segitiga
3
4
lingkaran
7
```

```
persegi
9
trapesium
8
4
3
```

Listing 1.7: luas.out

```
6.0
18.0
81.0
153.86
Total: 258.86
```

# Modul 2

# Struktur Data & Pemrograman Fungsional

## 2.1 Tujuan

Modul ini bertujuan untuk memberikan gambaran tentang Python dilihat dari paradigma pemrograman fungsional. Meski bukan didesain sejak awal seperti itu, tetapi ada beberapa komponen pemrograman fungsional yang diterapkan pada Python. Di samping itu, program ini juga mengenalkan beberapa struktur data bawaan dari Python yang dapat sangat memudahkan pengembangan purwarupa dari sebuah perangkat lunak.

Beberapa poin yang dipelajari antara lain:

1. Tipe data abstrak (ADT): *list*, *set*, *dictionary*, *stack*, *queue*
2. Paradigma pemrograman fungsional dan keuntungannya
3. Fungsi lambda dan *higher-order functions*
4. *Iterator* dan *generator*

## 2.2 Teori

### 2.2.1 Tipe Data Abstrak (ADT)

Dalam Python, dikenal beberapa ADT yang juga umum dipakai di bahasa lain, di antaranya: *list*, *set*, *dictionary*, *stack*, dan *queue*. *List* dan *dictionary* dalam Python bersifat *mutable*, sedangkan *set* atau *tuple* bersifat *immutable*. Perbandingan tiap ADT ini dalam Python menggunakan prinsip *lexicographical ordering*.

*List* dalam Python berfungsi seperti *array* dalam bahasa lain. Namun, terdapat beberapa metode bawaan dari ADT ini yang membuatnya bisa menjadi *stack* atau *queue*. Selain itu, *list* juga merupakan representasi dasar dari string dalam Python, yaitu dengan menjadikannya sebagai *list of characters*. *List* umumnya ditulis dengan menggunakan tanda kurung siku "[]".

*Dictionary* merupakan ADT yang berbentuk pasangan *key-value*. Dalam bahasa pemrograman lain, ADT ini lebih dikenal dengan nama *map* atau *hashmap*. *Dictionary* ditulis dengan menggunakan tanda kurung kurawal "{}".

### 2.2.2 Pemrograman Fungsional

Paradigma pemrograman fungsional menggunakan pendekatan deklaratif alih-alih imperatif seperti pada pemrograman prosedural. Setiap fungsi dibuat hanya bertanggung jawab terhadap satu operasi kecil. Oleh karena itu, paradigma pemrograman fungsional membuat fungsi pemrograman layaknya fungsi matematis. Ini membuat modularitas dari program yang dihasilkan bisa lebih tinggi dan mempermudah proses *debugging*.

Dalam paradigma pemrograman fungsional, salah satu konsep yang penting adalah keberadaan fungsi anonim atau fungsi lambda, dan *higher-order functions*. Fungsi lambda merupakan fungsi yang tidak memiliki identitas, i.e. tidak dideklarasikan nama fungsinya. Fungsi seperti ini biasanya dijadikan sebagai argumen untuk fungsi lain (*higher-order functions*). Beberapa contoh *higher-order functions* antara lain: `filter()`, `map()`, `max()`, `min()`, dan `sorted()`.

### 2.2.3 Iterator & Generator

*Iterator* digunakan seagai objek yang mengembalikan arus data, satu data setiap waktunya. *Iterator* juga dapat dilihat sebagai *singly-linked list* karena harus mempunyai metode `next()`. Sementara itu, *generator* merupakan fungsi spesial yang mempermudah pembuatan *iterator*. Dalam *generator*, nilai keluaran dari fungsi tersebut ditulis dengan kata kunci `yield`.

## 2.3 Praktikum

Listing 2.1: high.py

```
def f(x, g):
    return g(x) + 10

def kuadrat(x):
    return x ** 2

def kubik(x):
    return x ** 3

print f(3, kuadrat)
print f(3, kubik)
print f(3, lambda x: x ** 4)
```

## 2.4 Tugas Akhir

### 2.4.1 Deskripsi

**Minimum Scalar Product** (Google Code Jam Round 1A 2008)

Anda diberikan dua vektor $v_1 = (x_1, x_2, ..., x_n)$ dan $v_2 = (y_1, y_2, ..., y_n)$. Hasil perkalian produk dari dua vektor ini adalah sebuah nilai skalar yang dihitung sebagai $x_1y_1 + x_2y_2 + ... + x_ny_n$.

Asumsikan bahwa Anda diperkenankan untuk mengubah urutan dari elemen masing-masing vektor (permutasi). Pilihlah dua permutasi yang hasil perkalian produknya minimal, lalu munculkan hasil perkalian produk tersebut.

Baris pertama dari file masukan adalah $N$ jumlah kasus. Tiap kasus akan terdiri dari tiga baris yang terdiri dari $T$ jumlah elemen dari vektor pada baris pertama, dan dua baris berikutnya yang berisi dua vektor yang diberikan.

### 2.4.2 Contoh

Listing 2.2: A.in

```
2
3
1 -5 3
-2 1 4
5
5 4 3 1 2
1 1 0 1 0
```

Listing 2.3: A.out

```
Case #1: -25
Case #2: 6
```

### 2.4.3 Penjelasan

Berdasarkan kasus masukan di atas, $N = 2$. Kasus pertama, terdapat dua vektor dengan 3 elemen ($T = 3$), sedangkan pada kasus kedua terdapat dua vektor dengan 5 elemen ($T = 5$). Pada kasus pertama, permutasi yang menghasilkan produk skalar yang minimum adalah $(-5, 1, 3)$ dan $(4, 1, -2)$. Hasil perkalian produknya adalah $-5 \times 4 + 1 \times 1 + 3 \times -2 = -25$. Untuk kasus kedua, permutasi dan hasil produk skalarnya adalah $(5, 4, 3, 2, 1) \cdot (0, 0, 1, 1, 1) = 6$.

# Modul 3

# Pemrograman Berorientasi Objek

## 3.1 Tujuan

Pada Modul 2, kita sudah mengenal Python dalam paradigma pemrograman fungsional. Kali ini, kita akan melihat Python sebagai paradigma pemrograman berorientasi objek (OOP). Modul ini berfokus pada implementasi paradigma OOP dalam Python dan tidak banyak membahas dari sisi konsep OOP itu sendiri. Poin yang digarisbawahi pada modul ini adalah:

1. Modul, kelas, dan objek pada Python
2. Konsep *inheritance*
3. Proyek sederhana menggunakan Python
4. Penanganan error dan pengujian terotomasi

## 3.2 Teori

### 3.2.1 Pemrograman Berorientasi Objek

Dalam Python, kita bisa melihat modul sebagai kelas. Modul tersebut diwakili oleh *files*. Fungsi dan variabel yang tersimpan dalam modul tersebut dapat diakses seperti mengakses metode dan atribut pada suatu objek, yakni dengan operator dot. Meski begitu, Python juga memiliki cara untuk mendeklarasikan kelas dengan menggunakan kata kunci `class`.

Instansiasi kelas untuk menghasilkan objek dapat dilakukan dengan notasi pemanggilan fungsi, e.g. `c = Kelas()`. Dengan begitu, instansiasi objek dapat dilihat sebagai fungsi tanpa parameter yang dapat di-*assign* ke variabel. Hal ini dikenal juga secara generik dengan nama "konstruktor". Konstruktor didefinisikan dalam metode `__init__()`.

### 3.2.2 Penanganan Error

Ada dua jenis error secara umum: *syntax errors* dan *exceptions*. *Syntax errors* adalah kesalahan yang muncul karena tidak sesuai dengan kaidah penulisan dalam bahasa Python, e.g. salah indentasi, kurang tanda kurung, atau salah letak tanda koma. Di luar kesalahan

sintaksis tersebut, terdapat error yang muncul saat eksekusi program yang disebut sebagai *exceptions*.

Beberapa jenis error sudah ada dalam modul `exceptions`, contohnya:

1. `AssertionError`
2. `IOError`
3. `ImportError`
4. `KeyboardInterrupt`
5. `NotImplementedError`

Lengkapnya daftar jenis error tersebut bisa dilihat di `https://docs.python.org/2/library/exceptions.html\#bltin-exceptions`. Kita juga dapat mendefinisikan sendiri kelas *exception* baru jika diperlukan.

### 3.2.3 Pengujian

Python sudah memiliki modul untuk melakukan pengujian terotomasi pada level *unit testing*, i.e. `unittest`. Beberapa konsep penting dalam otomasi pengujian dengan `unittest`:

1. **test fixture**: persiapan dan pembersihan pengujian, e.g. membuat basis data temporer
2. **test case**: unit terkecil pengujian, memeriksa apakah keluarannya sudah sesuai dengan yang diharapkan dari masukannya
3. **test suite**: sekumpulan *test cases* yang perlu dijalankan bersamaan
4. **test runner**: komponen yang menjalankan dan memberikan hasilnya kepada pengguna, e.g. GUI atau CLI

Pengujian terotomasi ini melatih kita untuk mengetahui ekspektasi terhadap nilai keluaran dari fungsi tersebut dan mengujikannya dalam level terendah. Dengan demikian, kesalahan pada saat integrasi aplikasi dapat diminimalkan.

## 3.3 Praktikum

Listing 3.1: m.py

```
import math
import string

print "Latihan modul"
print math.pi
print math.sin(math.pi/2)
print math.sqrt(9)
```

```
print math.log(math.e)
print string.lowercase
print string.punctuation
```

Listing 3.2: fibo.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Listing 3.3: uji.py

```
import unittest

class TestStringMethods(unittest.TestCase):

    def setUp(self):
        self.foo = 'foo'
        self.bar = 'BAR'

    def test_upper(self):
        self.assertEqual(self.foo.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue(self.bar.isupper())
        self.assertFalse(self.foo.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

## 3.4 Tugas Akhir

Lihat kembali tugas akhir pada Modul 1! Berdasarkan soal tersebut,

1. Buatlah penyelesaian kasus tersebut dalam paradigma pemrograman berorientasi objek! Petunjuk: Anda perlu membuat kelas *parent* dengan nama `BangunDatar`. Kelas `BangunDatar` akan memiliki metode kosong bernama `hitung_luas()`.

2. Buatlah masing-masing tiga kasus uji untuk kuis di minggu pertama tersebut! Tiap-tiap kasus uji fungsi perhitungan luas bangun datar tersebut harus memiliki:

   (a) kasus nilai benar (`assertEqual()`)

   (b) kasus nilai salah (`assertNotEqual()`)

   (c) kasus tipe salah (`assertRaises(TypeError)`)

   Gunakan modul `unittest` dalam mengerjakan soal ini.

3. Buatlah deklarasi fungsi baru untuk menghitung luas segi enam! Namun, isi dari fungsi tersebut hanya mengeluarkan *exception* bahwa fungsi tersebut belum diimplementasi.

# Modul 4

# Pemrograman Web: Flask

## 4.1 Tujuan

Python merupakan bahasa pemrograman yang sangat *versatile*. Salah satu kegunaannya adalah untuk membentuk aplikasi web. Modul kali ini berfokus pada salah satu *framework* yang dapat digunakan untuk mengembangkan aplikasi web sederhana: Flask. Poin-poin yang akan difokuskan pada modul ini antara lain:

1. Penggunaan *virtual environments*;
2. *Routing* dan *templates*; serta
3. *Object-relational mapper* (ORM) dengan SQLAlchemy

## 4.2 Teori

Flask adalah sebuah contoh **microframework**, lawannya adalah *full-stack framework*, e.g. Django. Tidak banyak modul yang disediakan: tidak ada modul otentikasi, abstraksi basis data, atau sanitasi masukan. Oleh karena itu, Flask digunakan untuk mengerjakan aplikasi web yang cenderung sederhana .Beberapa contoh *microframeworks* untuk bahasa pemrograman lain di antaranya: Bottle untuk Python, Sinatra untuk Ruby, Lumen untuk PHP.

Flask menggunakan Jinja2 sebagai *templating engine*. Dengan demikian, penulisan file HTML dan kode Python dapat dipisah. Hal ini akan memudahkan pembuatan web dinamis dalam Flask.

Untuk memudahkan mengerjakan aplikasi dengan basis data di Flask, kita dapat menggunakan ekstensi SQLAlchemy. Pada dasarnya, setiap tabel dalam basis data dapat dianggap sebagai kelas. *Object-relational mapper* (ORM) berfungsi untuk **memetakan kelas ke basis data**. Dengan demikian **kolom pada tabel** dapat diakses seperti mengakses **atribut suatu kelas**. ORM juga memungkinkan untuk manipulasi antartabel, e.g. join, sehingga diharapkan dapat mengurangi tugas yang *redundant*.

## 4.3 Praktikum

Listing 4.1: log.py

```
from flask import Flask, session, redirect, url_for, escape, request

app = Flask(__name__)

app.route('/')def index():if 'username' in session:return 'Logged in as return 'You
    are not logged in'app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form method="post">
            <p><input type="text" name="username" placeholder="Username
               "></p>
            <p><input type="password" name="password" placeholder="
               Password"></p>
            <p><input type="submit" value="Login"></p>
        </form>
    '''

app.route('/logout')def logout():session.pop('username', None)return
    redirect('/')app.errorhandler(404)
def not_found(error):
    return 'Page not found!', 404

# set the secret key.  keep this really secret:
app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Listing 4.2: db.py

```
import datetime
import mysql.connector
import yaml

with open('config.yaml') as f:
    config = yaml.load(f)

cnx = mysql.connector.connect(**config)
cursor = cnx.cursor()

query = ("SELECT username, email, is_superuser, date_joined FROM
    auth_user "
         "WHERE date_joined BETWEEN %s AND %s")

date_start = datetime.date(2017, 1, 1)
date_end = datetime.date(2017, 3, 10)

cursor.execute(query, (date_start, date_end))
```

```
for row in cursor:
    print row

cursor.close()
cnx.close()
```

Listing 4.3: fs.py

```
from datetime import datetime
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:////tmp/test.db'
db = SQLAlchemy(app)


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)

    def __init__(self, username, email):
        self.username = username
        self.email = email

    def __repr__(self):
        return '<User %r>' % self.username

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(80))
    body = db.Column(db.Text)
    pub_date = db.Column(db.DateTime)

    category_id = db.Column(db.Integer, db.ForeignKey('category.id'))
    category = db.relationship('Category',
        backref=db.backref('posts', lazy='dynamic'))

    def __init__(self, title, body, category, pub_date=None):
        self.title = title
        self.body = body
        if pub_date is None:
            pub_date = datetime.utcnow()
        self.pub_date = pub_date
        self.category = category

    def __repr__(self):
        return '<Post %r>' % self.title

class Category(db.Model):
    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(50))

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return '<Category %r>' % self.name
```

## 4.4 Tugas Akhir

Menggunakan Flask, Anda diminta untuk membuat *urban dictionary*[1] untuk bahasa Indonesia. *Urban dictionary* adalah kamus dengan mekanisme *crowd sourcing*. Artinya, setiap pengguna diberikan kebebasan untuk mengisi lema dalam kamus tersebut beserta definisinya.

Sebagai contoh, kata "curcol" belum terdaftar dalam KBBI V. Seorang pengguna kemudian ingin memasukkan kata tersebut sebagai lema dalam kamus urban yang Anda buat. Pengguna tersebut hanya perlu *login* atau mendaftarkan diri jika belum terdaftar, lalu memasukkan kata "curcol" beserta definisinya.

Anda perlu membuat satu halaman lagi yang berisi indeks, i.e. daftar kata yang telah masuk ke dalam kamus urban tersebut. Anda dapat menggunakan parameter dari *routing* untuk membagi halaman indeks berdasarkan huruf awal dari kata tersebut. Perhatikan bahwa satu kata dapat memiliki lebih dari satu definisi.

Daftar kata dan definisinya, daftar pengguna, dan pengembangan lainnya harus Anda simpan dalam basis data menggunakan SQLAlchemy. Anda dapat menggunakan SQLAlchemy yang berupa ekstensi dari Flask. Untuk pengembangan lebih lanjut, sebagai bonus, Anda juga dapat menyimpan *voting* dari suatu definisi untuk menunjukkan bahwa definisi tersebut disukai/tidak disukai banyak orang.

[1] `http://www.urbandictionary.com/`

# Modul 5

# Pemrograman Web: Scraping

## 5.1 Tujuan

Praktikum kelima ini bertujuan untuk mengenalkan proses *web scraping* menggunakan pustaka Requests dan Beautiful Soup. Sebagai perbandingan, ditunjukkan juga proses *scraping* dengan menggunakan *framework* Scrapy.

## 5.2 Teori

Untuk membuat sebuah perangkat lunak, terkadang kita perlu menggunakan protokol atau kakas yang telah disediakan oleh orang lain. Sebagai contoh, kita perlu data yang sudah disediakan orang lain untuk dianalisis, e.g. analisis sentimen dari *tweets* di Twitter. Terdapat situs yang langsung menyediakan dataset, tetapi ada juga situs yang menyediakan Application Programming Interface (API).

Beberapa situs tidak menyediakan dataset atau API untuk memberikan datanya karena:

1. tidak dikembangkan sejak awal;
2. tidak ingin datanya disebarkan, e.g. Instagram; atau
3. hanya bisa diakses terbatas, e.g. Microdata BPS

sehingga mungkin perlu dilakukan *scraping*. Namun, perlu diketahui bahwa ada masalah etika yang harus diperhatikan. "visible $\neq$ accessible $\neq$ storable $\neq$ presentable" (Lavrenko, 2010).

Pustaka seperti Requests membantu kita dalam melakukan proses akses API atau *scraping* tersebut. Dengan metode yang cukup mudah digunakan, kita bisa mendapatkan berbagai macam *response content* dari web. Tidak hanya itu, Requests juga memudahkan kita jika dalam proses tersebut dibutuhkan otentikasi.

Pada prinsipnya, setiap hal yang terlihat di peramban web (*web browser*) bisa di-*scrape*. Yang perlu dilakukan hanya mengambil berkas HTML, lalu mengurainya (*parsing*). Beautiful Soup merupakan pustaka Python yang dapat membantu kita dalam mengurai file HTML tersebut. Dengan menggunakan cara akses melalui struktur data pohon, kita bisa mengambil elemen-elemen penting dari suatu halaman web.

## 5.3 Praktikum

Listing 5.1: req.py

```
import requests
import warnings
warnings.filterwarnings("ignore")

# Mencoba HTTP GET
params = {'q': 'Homer Simpson', 'format': 'json'}
r = requests.get('https://api.duckduckgo.com/', params=params)
print r.json()

# Mencoba HTTP POST
payload = {'status': 'Kok belajar Python susah amat ya?'}
r = requests.post('http://httpbin.org/post', data=payload)
print r.json()
```

Listing 5.2: bs.py

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(open('alice.html'), 'html.parser')

print soup.title
print soup.title.string
print soup.p
print soup.p['class']
print soup.find_all('a')
print soup.find(id='link3')
```

## 5.4 Tugas Akhir

Situs *What If?* dari `xkcd.com` merupakan situs yang menampilkan artikel-artikel yang berisi jawaban ilmiah untuk pertanyaan-pertanyaan hipotetis yang absurd. Secara umum, tiap halaman berisikan informasi judul, pertanyaan, penanya, jawaban, dan gambar pendukung. Tugas Anda adalah mengumpulkan data tersebut dari situs `https://what-if.xkcd.com/` dan menyimpannya dalam basis data.

Setiap gambar yang diambil disimpan dalam sebuah folder yang diberi nama sesuai dengan nomor artikel tersebut. Sebagai catatan, di setiap artikel selalu ada gambar *header* yang harus dikecualikan. Selain itu, beberapa artikel juga mengandung rumus/formula yang dapat diabaikan dari proses penyimpanan data.