



Laporan Praktikum Algoritma & Pemrograman

Semester Genap 2024/2025

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

NIM	<71231046>
Nama Lengkap	<FREIRE HANAN PUTRA>
Minggu ke / Materi	11 / DICTIONARY

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA
YOGYAKARTA
2025

BAGIAN 1: MATERI MINGGU INI (40%)

MATERI 1 Materi

Dictionary mempunyai kemiripan dengan list, tetapi lebih bersifat umum. Dalam list, indeks harus berupa integer sedangkan dalam dictionary indeks bisa dapat berupa apapun. Dictionary memiliki anggota yang terdiri dari pasangan kunci:nilai. Kunci harus bersifat unik, tidak boleh ada dua kunci yang sama dalam satu dictionary. Dictionary dapat diartikan pula sebagai pemetaan antara sekumpulan indeks (yang disebut kunci) dan sekumpulan nilai. Setiap kunci memetakan suatu nilai. Asosiasi kunci dan nilai tersebut disebut dengan pasangan nilai kunci (key-value pair) atau ada yang menyebutnya sebagai item. Sebagai contoh, kita akan mengembangkan kamus yang memetakan dari kata-kata dalam bahasa Inggris ke bahasa Spanyol, jadi bisa dikatakan yang menjadi kunci dan nilai adalah data string. Asumsinya setiap kata dalam bahasa Inggris mempunyai satu arti dalam bahasa Spanyol.

Fungsi dict digunakan untuk membuat dictionary kosong yang baru. Karena dict adalah fungsi bawaan (built-in function) dari Python, maka sebaiknya jangan digunakan sebagai nama variabel agar tidak menimbulkan konflik atau kebingungan dalam program.

```
>>> eng2sp = dict()
>>> print(eng2sp)
{}
```

Tanda kurung kurawal { } digunakan untuk membuat dictionary kosong. Untuk menambahkan item ke dalam dictionary, kita bisa menggunakan tanda kurung siku [] dengan menyebutkan kuncinya, lalu menetapkan nilainya.

```
>>> eng2sp['one'] = 'uno'
```

Jika kita perhatikan potongan kode tersebut, terlihat bahwa baris tersebut menambahkan item baru ke dalam dictionary, di mana kunci 'one' dipetakan ke nilai 'uno'. Saat dictionary tersebut dicetak, akan ditampilkan pasangan kunci dan nilai yang menunjukkan hubungan antara keduanya.

```
>>> print(eng2sp)
{'one': 'uno'}
```

Format output yang ditampilkan saat mencetak dictionary sama dengan format saat membuatnya. Misalnya, jika kita membuat dictionary baru dengan tiga item, lalu mencetaknya, maka hasil yang muncul akan menampilkan seluruh isi dictionary dalam bentuk pasangan kunci dan nilai, sesuai dengan struktur yang digunakan saat input.

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Urutan pasangan kunci-nilai dalam dictionary tidak selalu tetap atau teratur. Secara umum, kita tidak bisa memprediksi urutan item di dalam dictionary. Namun, hal ini bukanlah masalah besar karena dictionary tidak menggunakan indeks angka seperti list. Sebagai gantinya, kita menggunakan kunci untuk mengakses nilai yang sesuai, sehingga posisi item dalam dictionary tidak memengaruhi fungsinya.

```
>>> print(eng2sp['two'])
'dos'
```

Kunci 'two' selalu terhubung dengan nilai "dos", sehingga meskipun urutan item dalam dictionary berubah-ubah, hal tersebut tidak berpengaruh terhadap fungsinya. Yang penting adalah hubungan antara kunci dan nilainya tetap konsisten. Namun, jika kita mencoba mengakses kunci yang tidak ada dalam dictionary, maka Python akan menghasilkan error berupa pengecualian (KeyError).

```
>>> print(eng2sp['four'])
KeyError: 'four'
```

Fungsi **len** pada *dictionary* digunakan untuk mengembalikan jumlah pasangan nilai kunci:

```
>>> len(eng2sp)
3
```

Operator **in** dalam *dictionary* mengembalikan nilai benar (*true*) atau salah (*false*) sesuai dengan kunci yang ada dalam *dictionary*.

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

Untuk mengetahui nilai-nilai yang terdapat dalam sebuah dictionary, kita bisa menggunakan method `.values()`. Method ini akan mengembalikan semua nilai di dalam dictionary dalam bentuk objek khusus, yang bisa dikonversi menjadi list. Setelah dikonversi ke list, kita bisa menggunakan operator **in** untuk memeriksa apakah suatu nilai ada dalam dictionary tersebut.

```
>>> vals = list(eng2sp.values())
>>> 'uno' in vals
True
```

Operator **in** bekerja dengan cara yang berbeda pada list dan dictionary. Pada **list**, operator **in** menggunakan **algoritma pencarian linear**, artinya Python akan memeriksa setiap elemen satu per satu dari awal hingga akhir. Semakin panjang list-nya, semakin lama waktu yang dibutuhkan untuk menemukan (atau memastikan tidak ada) elemen yang dicari.

Sementara itu, pada **dictionary**, Python menggunakan struktur data yang disebut **Hash Table**. Dengan algoritma ini, pencarian menggunakan operator **in** menjadi sangat cepat dan **tidak tergantung pada jumlah item** dalam dictionary. Artinya, meskipun dictionary berisi ratusan atau ribuan item, waktu pencarian tetap **konstan** (disebut waktu konstan atau *constant time*), karena Python bisa langsung menuju lokasi data berdasarkan hash dari kuncinya.

MATERI 2 Dictionary Sebagai Set Penghitung (Counts)

Kita diberikan sebuah string dan ingin menghitung berapa kali setiap huruf muncul. Ada beberapa cara yang bisa digunakan:

1. Menggunakan 26 variabel terpisah

Membuat variabel untuk setiap huruf alfabet, lalu memeriksa setiap karakter dalam string menggunakan kondisi berantai (misalnya if atau elif) untuk menambah hitungan variabel yang sesuai.

2. Menggunakan list dengan 26 elemen

Membuat list dengan 26 slot, masing-masing mewakili satu huruf. Setiap karakter dalam string dikonversi menjadi angka (misalnya dengan fungsi bawaan seperti ord()), kemudian angka tersebut dipakai sebagai indeks untuk menambah nilai di list.

3. Menggunakan dictionary

Membuat dictionary yang kuncinya adalah karakter, dan nilainya adalah jumlah kemunculan karakter tersebut. Saat karakter ditemukan, kita tambahkan nilainya di dictionary. Jika karakter belum ada, kita buat dulu item baru dalam dictionary.

Meskipun ketiga cara di atas menghasilkan hasil yang sama, metode perhitungan atau implementasinya berbeda-beda. Di antara ketiganya, penggunaan **dictionary** biasanya lebih praktis dan efisien karena kita tidak perlu tahu huruf apa saja yang akan muncul sebelumnya. Kita cukup menambahkan item baru ke dictionary saat menemukan huruf baru, tanpa harus menyiapkan ruang atau variabel khusus untuk setiap huruf.

```
1 word = 'brontosaurus'
2 d = dict()
3 for c in word:
4     if c not in d:
5         d[c] = 1
6     else:
7         d[c] = d[c] + 1
8 print(d)
```

```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

Model komputasi yang dijelaskan tadi disebut **histogram**, yaitu sebuah istilah dalam statistika yang menggambarkan frekuensi atau jumlah kemunculan setiap elemen dalam sebuah data.

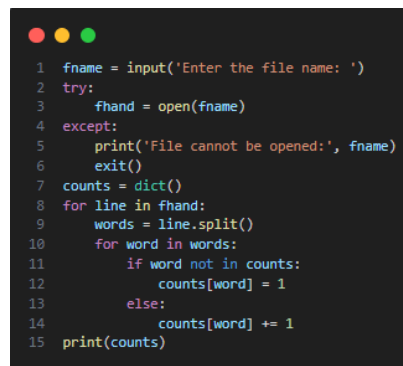
Pada prosesnya, perulangan for akan menelusuri setiap karakter dalam string. Jika karakter c belum ada dalam dictionary, maka dibuatlah item baru dengan kunci c dan nilai awal 1, yang berarti karakter tersebut sudah muncul sekali. Namun, jika karakter c sudah ada dalam dictionary, maka nilai pada kunci c tersebut akan langsung ditambahkan satu, menandakan kemunculan karakter tersebut bertambah satu kali.

Dictionary memiliki metode yang disebut *get* yang mengambil kunci dan nilai default. Jika kunci muncul di *dictionary*, akan mengembalikan nilai yang sesuai; jika tidak maka akan mengembalikan nilai default. Sebagai contoh:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

MATERI 3 Dictionary dan File

Salah satu kegunaan dictionary adalah untuk menghitung jumlah kemunculan kata kata pada sebuah file dengan beberapa teks tertulis. Kita akan membuat program Python yang membaca setiap baris dari file, memecahnya menjadi kata-kata, lalu menghitung frekuensi tiap kata menggunakan dictionary. Program ini menggunakan dua perulangan bersarang (nested loop): perulangan luar untuk membaca baris demi baris, dan perulangan dalam untuk memproses setiap kata dalam baris tersebut. Dengan cara ini, setiap kata di setiap baris akan dihitung secara menyeluruh.



```
1 fname = input('Enter the file name: ')
2 try:
3     fhand = open(fname)
4 except:
5     print('File cannot be opened:', fname)
6     exit()
7 counts = dict()
8 for line in fhand:
9     words = line.split()
10    for word in words:
11        if word not in counts:
12            counts[word] = 1
13        else:
14            counts[word] += 1
15 print(counts)
```

Pada statement else digunakan model penulisan yang lebih singkat untuk menambahkan variable. `counts[word] += 1` sama dengan `counts[word] = counts[word] + 1`. Metode lain dapat digunakan untuk mengubah nilai suatu variabel dengan jumlah yang diinginkan, misalnya dengan menggunakan `- =`, `* =`, dan `/ =`.

MATERI 4 Looping dan Dictionary

Dalam statement for, dictionary akan bekerja dengan cara menelusuri kunci yang ada didalamnya. Looping ini akan melakukan pecetakan setiap kunci sesuai dengan hubungan nilainya.

```
1 counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
2 for key in counts:
3     if counts[key] > 10 :
4         print(key, counts[key])
```

Saat melakukan looping for melalui kunci-kunci dalam dictionary, kita harus menggunakan operator indeks `[]` untuk mengakses nilai yang terkait dengan setiap kunci tersebut.

Untuk mencetak kunci dictionary secara alfabet, pertama-tama kita ambil semua kunci dictionary menggunakan method `.keys()` dan ubah menjadi list. Kemudian, kita urutkan list tersebut dengan `.sort()`.

Setelah itu, kita lakukan perulangan pada list yang sudah diurutkan, dan pada setiap iterasi kita cetak pasangan kunci dan nilai dari dictionary sesuai urutan alfabet.

```
1 counts = {'chuck': 1, 'annie': 42, 'jan': 100}
2 lst = list(counts.keys()) # Ambil kunci dan ubah ke list
3 lst.sort()               # Urutkan list secara alfabet
4 for key in lst:
5     print(key, counts[key]) # Cetak kunci dan nilai sesuai urutan
6
```

MATERI 5 Advanced Text Parsing

Memiliki fungsi `split` yang secara otomatis memisahkan kalimat menjadi kata-kata dengan menggunakan spasi sebagai pemisah. Misalnya, kata "soft!" dan "soft" dianggap berbeda karena tanda seru (!) ikut dihitung sebagai bagian kata, sehingga dalam dictionary keduanya akan tercatat sebagai entri yang berbeda. Hal yang sama berlaku untuk huruf kapital; kata "who" dan "Who" dianggap berbeda dan dihitung secara terpisah.

Untuk mengatasi perbedaan ini, kita bisa menggunakan metode string lain seperti `.lower()` untuk mengubah semua huruf menjadi huruf kecil, serta menghilangkan tanda baca menggunakan modul `string.punctuation` dan metode `.translate()`. Metode `.translate()` ini dianggap sangat efektif dan halus dalam menghapus atau mengganti karakter tertentu pada string.

```
line.translate(str.maketrans(fromstr, tostr, deletestr))
```

```
>>> import string
>>> string.punctuation
'!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~'
```

Penggunaan dalam program sebagai berikut:

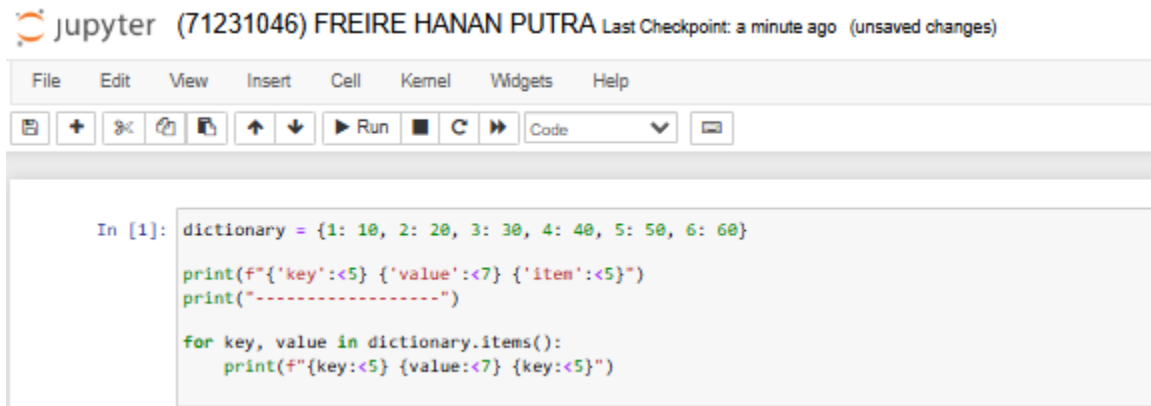
```
1 import string
2
3 fname = input('Enter the file name: ')
4 try:
5     fhand = open(fname)
6 except:
7     print('File cannot be opened:', fname)
8     exit()
9
10 counts = dict()
11 for line in fhand:
12     line = line.rstrip()
13     line = line.translate(line.maketrans('', '', string.punctuation))
14     line = line.lower()
15     words = line.split()
16     for word in words:
17         if word not in counts:
18             counts[word] = 1
19         else:
20             counts[word] += 1
21
22 print(counts)
```

BAGIAN 2: LATIHAN MANDIRI (60%)

Link Github: https://github.com/Freirehnn23/prak_alpro_week12

SOAL 1

Code:



Jupyter (71231046) FREIRE HANAN PUTRA Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run

```
In [1]: dictionary = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

print(f"{'key':<5} {'value':<7} {'item':<5}")
print("-----")

for key, value in dictionary.items():
    print(f"{'key':<5} {'value':<7} {'key':<5}")
```

Penerapan:


key	value	item
1	10	1
2	20	2
3	30	3
4	40	4
5	50	5
6	60	6

Penjelasan:













- `dictionary = {...}`: Membuat dictionary berisi pasangan key-value.
- `print(f"...")`: Mencetak header tabel dengan format rata kiri dan lebar tetap.
- `print("-----")`: Garis pemisah tabel.
- `for key, value in dictionary.items():`: Looping semua pasangan key-value.
- `print(f"...")`: Mencetak isi tabel dengan format kolom key, value, dan key lagi.
- Kolom item menampilkan ulang key, bukan pasangan (key, value).

SOAL 2

Code:

 jupyter (71231046) FREIRE HANAN PUTRA Last Checkpoint: 4 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

        Run    Code 

```
In [2]: Lista = ['red', 'green', 'blue']
        Listb = ['#FF0000', '#008000', '#0000FF']

        hasil = {}

        for i in range(len(Lista)):
            hasil[Lista[i]] = Listb[i]

        print(hasil)
```

Penerapan:

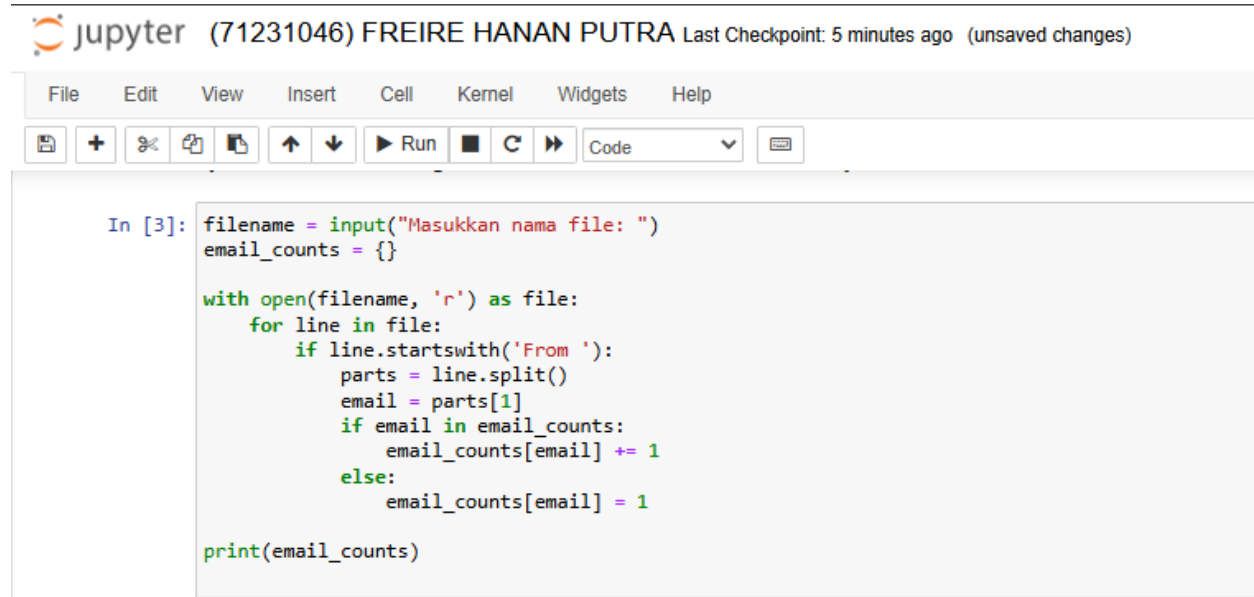
```
{'red': '#FF0000', 'green': '#008000', 'blue': '#0000FF'}
```

Penjelasan:

- Lista dan Listb: Dua list berisi nama warna dan kode heksadesimalnya.
- hasil = {}: Membuat dictionary kosong.
- for i in range(len(Lista)): Loop berdasarkan indeks dari list.
- hasil[Lista[i]] = Listb[i]: Menambahkan pasangan key-value ke dictionary.
 - Key = nama warna (Lista[i])
 - Value = kode warna (Listb[i])
- print(hasil): Menampilkan dictionary hasil mapping warna ke kodenya.

SOAL 3

Code:



```
In [3]: filename = input("Masukkan nama file: ")
email_counts = {}

with open(filename, 'r') as file:
    for line in file:
        if line.startswith('From '):
            parts = line.split()
            email = parts[1]
            if email in email_counts:
                email_counts[email] += 1
            else:
                email_counts[email] = 1

print(email_counts)
```

Penerapan:

Masukkan nama file: mbox-short.txt


```
{'stephen.marquard@uct.ac.za': 2, 'louis@media.berkeley.edu': 3, 'zqian@umich.edu': 4, 'rjlowe@iupui.edu': 2, 'cwen@iupui.edu': 5, 'gsilver@umich.edu': 3, 'wagnermr@iupui.edu': 1, 'antranig@caret.cam.ac.uk': 1, 'gopal.ramasammycook@gmail.com': 1, 'david.horwitz@uct.ac.za': 4, 'ray@media.berkeley.edu': 1}
```

Penjelasan:

- filename = input(...): Meminta nama file dari pengguna.
- email_counts = {}: Inisialisasi dictionary kosong untuk menghitung email.
- with open(...): Membuka file untuk dibaca.
- for line in file: Membaca file baris per baris.
- if line.startswith('From '): Memproses hanya baris yang diawali 'From ' (bukan 'From:').
- parts = line.split(): Memecah baris menjadi list kata.
- email = parts[1]: Mengambil email yang berada di indeks ke-1.
- if email in email_counts: Cek apakah email sudah ada di dictionary.
 - Jika ya → tambahkan jumlahnya.
 - Jika tidak → inisialisasi dengan 1.
- print(email_counts): Menampilkan jumlah kemunculan tiap email.

SOAL 4

Code:

 jupyter (71231046) FREIRE HANAN PUTRA Last Checkpoint: 6 minutes ago (unsaved changes)

```
File Edit View Insert Cell Kernel Widgets Help

[Save] [Run] [Stop] [Restart] [Code] [Output]

In [4]: filename = input("Masukkan nama file: ")
        domain_counts = {}

        with open(filename, 'r') as file:
            for line in file:
                if line.startswith('From '):
                    parts = line.split()
                    email = parts[1]
                    domain = email.split('@')[1]
                    if domain in domain_counts:
                        domain_counts[domain] += 1
                    else:
                        domain_counts[domain] = 1

        print(domain_counts)
```

Penerapan:

```
Masukkan nama file: mbox-short.txt
{'uct.ac.za': 6, 'media.berkeley.edu': 4, 'umich.edu': 7, 'iupui.edu': 8, 'caret.cam.ac.uk': 1, 'gmail.com': 1}
```

Penjelasan:

- filename = input(...): Meminta nama file dari pengguna.
- domain_counts = {}: Dictionary untuk menghitung jumlah domain email.
- with open(...): Membuka file untuk dibaca.
- for line in file: Membaca file baris per baris.
- if line.startswith('From '): Hanya memproses baris yang diawali 'From '.
- parts = line.split(): Memecah baris jadi list kata.
- email = parts[1]: Mengambil email dari indeks ke-1.
- domain = email.split('@')[1]: Mengambil domain dari email.
- Cek apakah domain sudah ada di domain_counts:
 - Jika ada: tambah 1.
 - Jika tidak: inisialisasi dengan 1.
- print(domain_counts): Menampilkan jumlah kemunculan setiap domain.