Laporan Praktikum Algoritma dan Pemrograman

Semester Genap 2023/2024

NIM	**<71230998>**
Nama Lengkap	**<MERLYANA HELENA PATRICIA>**
Minggu ke / Materi	**11/Dictionary**

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.

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS KRISTEN DUTA WACANA

YOGYAKARTA

2024

BAGIAN 1: MATERI MINGGU INI (40%)

MATERI

Dictionary?

Dalam praktikum kali ini, kita akan membahas tentang dictionary dalam Python. Dictionary mirip dengan list, tetapi lebih umum. Di dalam list, kita menggunakan indeks berupa integer, sedangkan dalam dictionary, indeks bisa berupa apa saja. Dictionary terdiri dari pasangan kunci:nilai, di mana kunci harus unik, artinya tidak boleh ada dua kunci yang sama dalam satu dictionary.

Kita bisa memahami dictionary sebagai sebuah pemetaan antara sekumpulan kunci dan sekumpulan nilai. Setiap kunci memetakan suatu nilai. Sebagai contoh, kita bisa membuat kamus yang memetakan kata-kata dalam bahasa Inggris ke bahasa Spanyol, di mana kata-kata Inggris menjadi kunci dan kata-kata Spanyol menjadi nilainya.

Untuk membuat dictionary baru, kita bisa menggunakan fungsi dict(), namun kita perlu hindari menggunakan nama variabel yang sama dengan fungsi bawaan Python. Kita bisa membuat dictionary kosong menggunakan tanda kurung kurawal {}. Misalnya, jika kita ingin menambahkan item ke dalam dictionary, kita bisa menggunakan kurung kotak []. Misalnya:

```
eng2sp = dict()
eng2sp['one'] = 'uno'
print(eng2sp)
```

Ini akan membuat item yang memetakan kunci 'one' ke nilai 'uno'. Hasilnya akan terlihat seperti {'one':'uno'}.

Kita juga bisa membuat dictionary dengan item-item sekaligus:

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

Penting untuk dicatat bahwa urutan pasangan kunci-nilai tidaklah tetap. Namun, kita bisa menggunakan kunci untuk mencari nilainya. Misalnya:

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

Ini akan menghasilkan 'dos', karena kunci 'two' selalu dipasangkan dengan nilai 'dos'.

Jika kita menggunakan kunci yang tidak ada dalam dictionary, akan muncul pengecualian:

```
print(eng2sp['four'])
```

Ini akan menghasilkan KeyError: 'four'.

Fungsi len() pada dictionary digunakan untuk mengembalikan jumlah pasangan kunci:nilai, dan operator 'in' digunakan untuk memeriksa keberadaan kunci dalam dictionary.

```
print(len(eng2sp))
print('one' in eng2sp)
print('uno' in eng2sp)
```

Selain itu, kita bisa menggunakan method values() untuk mendapatkan semua nilai dalam dictionary. Method ini akan mengembalikan nilai dalam bentuk list yang bisa kita gunakan dalam operator 'in'.

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

Operator 'in' menggunakan algoritma yang berbeda untuk list dan dictionary. Dalam dictionary, Python menggunakan algoritma yang disebut Hash Table, sehingga waktu pencarian tidak terpengaruh oleh jumlah item dalam dictionary.

Dictionary sebagai set penghitung (counters)

Sebagai lanjutan dari materi sebelumnya, kita akan membahas pendekatan yang lebih efisien untuk menghitung jumlah kemunculan huruf dalam sebuah string. Sebelumnya, kita telah membahas tiga cara untuk melakukan hal ini, dan kita telah melihat bahwa pendekatan dengan menggunakan dictionary adalah yang paling praktis.

Dictionary dalam Python memungkinkan kita untuk memetakan kunci (dalam hal ini, huruf) ke nilai (jumlah kemunculan). Dengan menggunakan dictionary, kita tidak perlu mendefinisikan variabel

atau list terpisah untuk setiap huruf atau melakukan konversi huruf menjadi angka untuk digunakan sebagai indeks.

Pendekatan dengan menggunakan dictionary dapat diimplementasikan dengan model histogram. Model histogram adalah istilah statistika yang merujuk pada set perhitungan atau frekuensi. Dalam model ini, kita menggunakan sebuah dictionary untuk menyimpan jumlah kemunculan setiap karakter. Mari kita lihat contoh implementasinya dengan string 'brontosaurus':

```
word = 'brontosaurus'
d = dict()

for c in word:
    if c not in d:
        d[c] = 1
    else:
        d[c] += 1

print(d)
```

Hasilnya akan sama dengan sebelumnya:

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

Kita juga bisa menggunakan metode get pada dictionary untuk menyederhanakan loop penghitungan ini. Metode get secara otomatis menangani kasus di mana kunci tidak ada dalam dictionary, sehingga kita tidak perlu menggunakan kondisional berantai untuk menambah nilai.

```
word = 'brontosaurus'
d = dict()

for c in word:
    d[c] = d.get(c, 0) + 1

print(d)
```

Hasilnya akan sama dengan pendekatan sebelumnya:

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

Dengan menggunakan metode get, kita dapat merampingkan kode menjadi lebih singkat dan lebih mudah dipahami. Pendekatan ini merupakan "idiom" yang sangat umum digunakan dalam Python untuk menghitung jumlah kemunculan elemen dalam sebuah sequence.

Dictionary dan File

Kita akan mempelajari penggunaan dictionary untuk menghitung kemunculan kata-kata dalam sebuah file teks. Kita akan mulai dengan file teks yang sangat sederhana yang diambil dari teks Romeo dan Juliet.

Pertama-tama, kita perlu menulis program Python yang membaca baris-baris file, memecah setiap baris menjadi daftar kata, dan kemudian melakukan perulangan melalui setiap kata dalam baris tersebut untuk menghitung kemunculan masing-masing kata menggunakan dictionary.

Pertama, kita akan meminta pengguna untuk memasukkan nama file. Kemudian, kita membuka file tersebut dan melakukan perulangan melalui setiap baris di dalamnya. Dalam setiap baris, kita memecah kata-kata menggunakan metode split() dan melakukan perulangan melalui setiap kata. Selama perulangan, kita memeriksa apakah kata tersebut sudah ada dalam dictionary counts. Jika belum, kita tambahkan kata tersebut ke dalam dictionary dengan nilai awal 1. Jika sudah ada, kita tambahkan 1 pada jumlah kemunculan kata tersebut.

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()

counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)
```

Dalam statement else, kita menggunakan model penulisan yang lebih singkat dengan menggunakan counts[word] += 1, yang sama dengan counts[word] = counts[word] + 1. Ini

memungkinkan kita untuk mengubah nilai suatu variabel dengan jumlah yang diinginkan dengan lebih ringkas.

Ketika program dijalankan dengan file teks 'romeo.txt', kita akan mendapatkan output berupa dictionary yang berisi jumlah kemunculan setiap kata dalam file tersebut. Outputnya akan berupa data dump dalam urutan hash yang tidak disortir. Jadi, dengan menggunakan nested loop, kita dapat menghitung jumlah kemunculan setiap kata dalam sebuah file teks dengan menggunakan dictionary sebagai wadah untuk hasil perhitungan tersebut.

Looping dan Dictionary

Dalam materi sebelumnya, kita telah mempelajari penggunaan dictionary untuk menghitung kemunculan kata-kata dalam sebuah file teks. Sekarang, kita akan melanjutkan dengan memahami cara menggunakan dictionary dalam looping dan pengurutan.

Dalam sebuah statement for yang mengiterasi melalui dictionary, kita akan mengakses kunci-kunci yang ada di dalamnya. Looping ini akan mencetak setiap kunci bersama dengan nilai yang sesuai. Outputnya tidak berurutan, karena dictionary tidak mempertahankan urutan tertentu. Sebagai contoh, jika kita punya dictionary seperti ini:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
```

dan kita melakukan looping seperti ini:

```
for key in counts:
    print(key, counts[key])
```

Outputnya akan seperti ini:

```
jan 100
chuck 1
annie 42
```

Untuk mencetak kunci dengan nilai di atas 10, kita tambahkan kondisi di dalam looping:

```
for key in counts:
    if counts[key] > 10 :
        print(key, counts[key])
```

Outputnya akan menampilkan data dengan nilai di atas 10 saja.

Untuk mencetak kunci dalam urutan alfabet, langkah pertama adalah membuat list dari kunci-kunci pada dictionary dengan menggunakan method keys(). Kemudian, kita melakukan pengurutan list tersebut dan melakukan looping melalui list yang sudah diurutkan untuk mencetak pasangan nilai kunci yang sudah diurutkan. Sebagai contoh:

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
lst.sort()
for key in lst:
    print(key, counts[key])
```

Outputnya akan seperti ini:

```
annie 42
chuck 1
jan 100
```

Pertama-tama, kita melihat list dari kunci-kunci yang tidak berurutan yang didapatkan dari method keys(). Kemudian kita melihat pasangan nilai kunci yang sudah diurutkan menggunakan loop for. Dengan cara ini, kita bisa mencetak nilai dalam urutan yang kita inginkan.

Advenced Text Parsing

Dalam materi sebelumnya, kita telah menggunakan contoh file teks yang telah dihilangkan tanda bacanya. Sekarang, kita akan menggunakan file teks yang mengandung tanda baca lengkap. Misalnya, kita akan menggunakan contoh yang sama yaitu romeo.txt dengan tanda baca lengkap:

```
But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
```

Ketika menggunakan dictionary untuk menghitung kemunculan kata-kata dalam teks dengan tanda baca, kita perlu memperlakukan kata-kata yang sama dengan bentuk yang berbeda (misalnya, dengan atau tanpa tanda baca) sebagai kata yang berbeda dalam dictionary.

Python memiliki fungsi split() yang akan membagi kata-kata berdasarkan spasi dan menganggap setiap kata sebagai token yang dipisahkan oleh spasi. Namun, fungsi ini tidak memperhatikan tanda baca yang melekat pada kata-kata. Misalnya, "soft!" dan "soft" akan dianggap sebagai dua kata yang berbeda.

Untuk menangani hal ini, kita dapat menggunakan method translate() dalam kombinasi dengan string.punctuation dari modul string. Method translate() digunakan untuk mengganti karakter yang sesuai dari satu string ke string lain, dan dalam hal ini kita akan menggunakan string.punctuation untuk menghapus semua tanda baca dari teks.

Berikut adalah implementasinya:

```
import string

fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()

counts = dict()
for line in fhand:
    line = line.rstrip()
    line = line.translate(line.maketrans('', '', string.punctuation))
    line = line.lower()  # Mengubah semua huruf menjadi huruf kecil
    words = line.split()  # Memecah baris menjadi kata-kata
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)
```

Output dari program di atas akan menghitung kemunculan setiap kata dalam teks 'romeo-full.txt', setelah tanda baca dihapus dan setiap kata diubah menjadi huruf kecil.

BAGIAN 2: LATIHAN MANDIRI (60%)

Link Github : https://github.com/memel25/71230998-pertemuan11.git

SOAL 10.1

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

print("key", "value", "item", sep='\t')

for key, value in dictionary.items():
    print(key, value, key, sep='\t')
```

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

Kita memiliki sebuah dictionary dengan kunci dan nilai tertentu. Kita ingin mencetak setiap pasangan kunci dan nilai, dan juga kunci tersebut sebagai 'item'. Untuk melakukannya, pertama-tama kita mencetak judul kolom 'key', 'value', 'item' dengan menggunakan `print("key", "value", "item", sep='\t')`. Kemudian, kita melakukan looping melalui setiap pasangan kunci dan nilai dalam dictionary menggunakan `for key, value in dictionary.items():`. Dalam setiap iterasi, kita mencetak kunci, nilai, dan kunci lagi sebagai 'item' dengan menggunakan `print(key, value, key, sep='\t')`. Dengan langkah-langkah ini, kita dapat mencetak dictionary dalam format yang diinginkan dengan kunci, nilai, dan kunci sebagai 'item'.

SOAL 10.2

```
list1 = []
while True:
    isi = input('masukan isi list / ketik "selesai" untuk lanjut ke list 2 = ')
    if isi.lower() == 'selesai':
        break
    else:
        list1.append(isi)
list2 = []
while True:
    isi2 = input('masukan isi list / ketik "selesai" untuk seelsai = ')
    if isi2.lower() == 'selesai':
        break
    else:
        list2.append(isi2)

result_dict = dict(zip(list1, list2))
print (result_dict)
```

```
masukan isi list / ketik "selesai" untuk lanjut ke list 2 = red
masukan isi list / ketik "selesai" untuk lanjut ke list 2 = green
masukan isi list / ketik "selesai" untuk lanjut ke list 2 = blue
masukan isi list / ketik "selesai" untuk lanjut ke list 2 = selesai
masukan isi list / ketik "selesai" untuk seelsai = #FF0000
masukan isi list / ketik "selesai" untuk seelsai = #008000
masukan isi list / ketik "selesai" untuk seelsai = #0000FF
masukan isi list / ketik "selesai" untuk seelsai = selesai
{'red': '#FF0000', 'green': '#008000', 'blue': '#0000FF'}
```

Pertama, pengguna diminta untuk memasukkan elemen-elemen ke dalam `list1`. Setelah selesai, pengguna diminta untuk memasukkan elemen ke dalam `list2`. Setelah kedua list terisi atau jika pengguna memasukkan "selesai" untuk kedua list, program menggabungkan elemen-elemen dari `list1` sebagai kunci dan elemen-elemen dari `list2` sebagai nilai dalam sebuah dictionary menggunakan fungsi `zip()`. Terakhir, program mencetak dictionary hasilnya.

SOAL 10.3

```
n_f = "mbox-short.txt"
file = open('mbox-short.txt', 'r')

h_e = {}
for i in file:
    if i.startswith('From '):
        kata = i.split()
        email = kata[1]
        h_e[email] = h_e.get(email, 0) + 1
file.close()

print(h_e)
```

Kode ini membuka file `'mbox-short.txt'` untuk dibaca. Kita membuat sebuah dictionary kosong bernama `h_e` yang akan digunakan untuk menghitung jumlah email dari setiap pengirim. Selanjutnya, kita membaca file baris per baris. Kita hanya tertarik pada baris-baris yang dimulai dengan 'From ', yang

menandakan header email. Ketika menemukan baris yang dimulai dengan 'From ', kita membaginya menjadi kata-kata dan mengambil alamat email pengirimnya.

Kemudian, kita memperbarui dictionary `h_e` dengan jumlah email dari setiap pengirim. Jika alamat email sudah ada dalam dictionary, kita tambahkan 1 ke nilai yang ada. Jika tidak, kita set nilai awalnya ke 1. Setelah selesai membaca seluruh file, kita menutup file tersebut.

Terakhir, kita mencetak dictionary `h_e` yang berisi jumlah email dari setiap pengirim. Dengan kode ini, kita dapat dengan mudah menghitung dan mencetak histogram jumlah email dari setiap pengirim dalam file `'mbox-short.txt'`.SOAL 10.4

SOAL 10.4

```
n_f = "mbox-short.txt"
file = open('mbox-short.txt', 'r')

hitung = {}
for i in file:
    if i.startswith('From '):
        kata = i.split()
        email = kata[1]
        hm = email.find('@')
        ha = email[hm + 1:]
        hitung[ha] = hitung.get(ha, 0)+1
file.close()

print(hitung)
```

Kode ini membuka file `'mbox-short.txt'` untuk dibaca. Kemudian, kita membuat sebuah dictionary kosong bernama `hitung` yang akan digunakan untuk menghitung jumlah email dari setiap domain. Selanjutnya, kita membaca file baris per baris. Kita hanya tertarik pada baris-baris yang dimulai dengan 'From ', yang menandakan header email.

Ketika menemukan baris yang dimulai dengan 'From ', kita membaginya menjadi kata-kata dan mengambil alamat email pengirimnya. Selanjutnya, kita mencari posisi '@' dalam alamat email untuk menemukan domain dari email tersebut. Dengan mengambil substring mulai dari posisi tersebut, kita bisa mendapatkan domain dari alamat email.

Kemudian, kita memperbarui dictionary `hitung` dengan jumlah email dari setiap domain. Jika domain sudah ada dalam dictionary, kita tambahkan 1 ke nilai yang ada. Jika tidak, kita set nilai awalnya ke 1. Setelah selesai membaca seluruh file, kita menutup file tersebut.

Terakhir, kita mencetak dictionary `hitung` yang berisi jumlah email dari setiap domain. Dengan kode ini, kita dapat dengan mudah menghitung dan mencetak histogram jumlah email dari setiap domain dalam file `'mbox-short.txt'`.