1. Input/ Output dan Operasi File

1.1 Input dari Keyboard

Program yang selama ini kita buat, nilainya telah ditentukan sebelumnya, program - program tersebut tidak mendapatkan input dari user. Program - program tersebut hanya melakukan hal yang sama setiap waktu.

Python menyediakan fungsi built-in yang mengambil nilai langsung dari input keyboard. Fungsi yang paling sederhana dinamakan raw_input . Ketika fungsi ini dipanggil, program dihentikan dan menunggu masukan dari user untuk mengetikkan sesuatu. Pada saat user menekan tombol [Enter<-|], maka program tersebut dilanjutkan dan fungsi raw_input mengembalikan apa yang user ketik, sebagai tipe data string, Contoh :

```
>>> input = raw_input()
silahkan ketik disini
>>> print input
silahkan ketik disini
```

Sebelum memanggil fungsi raw_input , sebaiknya kita memasukkan parameter argumen sebagai prompt pada raw_input :

```
>>> nama = raw_input ("Siapa nama anda?")
Siapa nama anda?santi
>>> print nama
santi
```

Jika kita mengharapkan input yang akan dimasukan bertipe data integer, kita dapat menggunakan fungsi input :

```
>>> prompt = "Siapa nama anda?"
>>> input (prompt)
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
TypeError: object of type 'string' is not callable
```

Tetapi jika yang dimasukan adalah digit angka atau bilangan maka secara otomatis akan dirubah menjadi tipe data integer, pada contoh di atas yang menjadi parameter dari input bukan berupa bilangan atau integer, maka program akan terhenti dan menampilkan pesan kesalahan. Untuk menghindarinya, sangat disarankan untuk menggunakan fungsi raw_input untuk mengambil nilai dengan tipe data string, kemudian menggunakan fungsi peubah tipe data untuk diubah menjadi tipe data lain yang diinginkan.

Pada saat suatu program berjalan, data tersimpan dalam memori. Kemudian pada saat program berhenti, atau komputer dimatikan, semua data yang tersimpan di memori akan hilang. Untuk menyimpan sebuah data secara permanen, Anda harus meletakan file tersebut ke dalam media penyimpanan, seperti Hard-Drive, CD-ROM, atau floppy disk.

Ketika file - file mencapai jumlah yang banyak, biasanya file - file tersebut dipisah - pisah dan di simpan ke dalam direktori - direktori. Masing - masing file

teridentifikasi dengan nomor yang unik, atau dengan kombinasi nama dan tempat direktori file masing-masing.

Mengoperasikan file sama halnya dengan mengoperasikan sebuah buku. Untuk membaca buku, Anda harus membukanya terlebih dahulu. Dan jika Anda selesai dengan buku tersebut Anda harus menutupnya kembali. Ketika suatu buku dalam keadaan terbuka, buku dapat ditulisi dan di baca. Operasi tersebut berlaku juga pada file.

1.2 Membuka File

Membuka sebuah file sama halnya dengan membuat sebuah objek file. Contoh berikut variabel f adalah objek file.

```
>>> f = open ("bulan.py", "r")
>>> type(f)
<type 'file'>
>>> print f
<open file 'bulan.py', mode 'r' at 0x8134fb0>
```

fungsi open, memerlukan 2 argumen parameter. argumen pertama adalah nama file yang akan dibuka, dan kedua adalah mode pembukaan file, contoh diatas file bulan.py dibuka dengan metode r yang berarti hanya dapat dibaca dan tidak bisa ditulisi. Dan jika kita membuka file yang tidak ada, akan muncul pesan kesalahan,
seperti berikut :

```
>>> f = open ("test", "r")
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
IOError: [Errno 2] No such file or directory: 'test'
```

1. metode r, hanya dapat ditulisi jika file tidak ada, akan muncul pesan kesalahan seperti contoh di atas
2. metode w, membuat suatu file baru hanya untuk di tulisi, jika terdapat nama file yang sama dalam suatu direktori maka file yang baru akan menimpa file yang lama.
3. metode a, ditambah dan setiap data yang akan ditulis akan diletakkan pada akhir file.
4. metode r+, dapat dibaca dan ditulisi.

1.3 Metode pada objek File

Untuk mengetahui lebih jelasnya tentang metode - metode yang dapat digunakan pada suatu objek file, kita dapat menggunakan fungsi dir(). Contohnya :

```
>>> dir(f)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__',
'__getattribute__', '__hash__', '__init__', '__iter__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__',
'close', 'closed', 'encoding', 'fileno', 'flush', 'isatty', 'mode',
'name', 'newlines', 'next', 'read', 'readinto', 'readline',
'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write',
'writelines', 'xreadlines']
```

* metode read(), menampilkan seluruh isi suatu file dengan tipe data string. M isalnya :

```
>>> f.read()
'# fungsi mulai di sini\ndef swap(x, y):\n    print "Dalam fungsi:"\n
print "\\tSebelum proses:"\n    print "\\t\\tNilai x", x\n    print
"\\t\\tNilai y", y\n    z = x\n    x = y\n    y = z\n    print
"\\tSetelah proses:"\n    print "\\t\\tNilai x", x\n    print
"\\t\\tNilai y", y\n\n# program utama mulai di sini\nx = 12\ny =
3\nprint "Sebelum memanggil fungsi, x bernilai", x\nprint "Sebelum
memanggil fungsi, y bernilai", y\nswap(x,y)\nprint "Setelah memanggil
fungsi, x bernilai", x\nprint "Setelah memanggil fungsi, y bernilai",
y\n'
```

* metode read(), menelusuri isi file sampai akhir file dan apabila metode read() sudah mencapai akhir dari file (atau sudah menggunakan metode read() sebelumnya).

```
>>> f.read()
''
```

* metode f.readline(), membaca isi file perbaris atau sampai menemukan karakter ASCII "\n". Sama seperti halnya penggunaan read(). Metode readline() akan menampilkan string kosong jika sudah mencapai akhir dari file. Contohnya :

```
>>> f.readline()
'# fungsi mulai di sini\n'

>>> f.readline()
'def swap(x, y):\n'

>>> f.readline()
'    print "Dalam fungsi:"\n'

>>> f.readline()
'    print "\\tSebelum proses:"\n'

>>> f.readline()
'    print "\\t\\tNilai x", x\n'

>>> f.readline()
'    print "\\t\\tNilai y", y\n'

>>> f.readline()
'    z = x\n'

>>> f.readline()
'    x = y\n'

>>> f.readline()
'    y = z\n'

>>> f.readline()
'    print "\\tSetelah proses:"\n'

>>> f.readline()
'    print "\\t\\tNilai x", x\n'
```

```
>>> f.readline()
'    print "\\t\\tNilai y", y\n'

>>> f.readline()
'\n'
>>> f.readline()
'# program utama mulai di sini\n'

>>> f.readline()
'x = 12\n'

>>> f.readline()
'y = 3\n'

>>> f.readline()
'print "Sebelum memanggil fungsi, x bernilai", x\n'

>>> f.readline()
'print "Sebelum memanggil fungsi, y bernilai", y\n'

>>> f.readline()
'swap(x,y)\n'

>>> f.readline()
print "Setelah memanggil fungsi, x bernilai", x\n'

>>> f.readline()
'print "Setelah memanggil fungsi, y bernilai", y\n'

>>> f.readline()
''
```

* metode write(<string>), memerlukan argumen parameter string, untuk dituliskan ke dalam file dan tidak menghasilkan tampilan hasil. Perhatikan metode file yang digunakan, metode r akan menimbulkan pesan kesalahan jika file dibuka dengan metode write.

```
>>> f = open ("tes", "w")
>>> f.write("Tes untuk menulis ke dalam file")
>>> w = open ("bulan.py", "r")
>>> w.write("TES")
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
IOError: [Errno 9] Bad file descriptor
```

* metode seek(<offset>, <dari>), metode ini akan melihat isi byte pada file pada posisi tertentu. Parameter <dari> jika bernilai 0 maka penelusuran byte dimulai dari awal file, jika bernilai 1 maka pada posisi byte yang sekarang(current position), dan jika bernilai 2 maka dimulai pada posisi byte akhir dari file.

```
>>> f = open ("tes.dat", "r+")
>>> f.write('0123456789')
>>> f.seek(2) #Menuju byte ke dua dari awal file
```

```
>>> f.read(1)
'2'
>>> f.seek(-3, 2) #Dimulai dari akhir file, baca 3 byte kebelakang.
>>> f.read(1)
'7'
```

* Jika Anda selesai menggunakan file tersebut, Anda dapat menggunakan metode close() untuk menghilangkannya dari memori, setelah melakukan metode close() maka secara otomatis penggunaan kembali file tersebut akan menampilkan pesan kesalahan.

```
>>> f.close()
>>> f.write()
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
ValueError: I/O operation on closed file
```

2. Errors dan Exception

2.1 Penelusuran program

Pemograman komputer adalah sebuah proses yang sangat kompleks, dan karena hal ini dilakukan manusia maka seringkali terjadi kesalahan - kesalahan. Pada umumnya kesalahan - kesalahan dalam pemograman dikenal sebagai bugs dan proses penelusuran program kembali dan memperbaikinya dikenal dengan debugging.

Tiga jenis kesalahan dapat terjadi dalam sebuah program: Kesalahan sintaks, kesalahan pada saat menjalankan program, yang kemudian dikenal sebagai runtime errors dan kesalahan algoritma program, yang kemudian dikenal dengan semantic errors. Sangatlah berguna untuk membedakannya satu sama lain dengan tujuan untuk mendeteksi kesalahan dan memperbaikinya dengan segera.

2.1.1 Kesalahan sintaks

Python hanya dapat mengeksekusi program tersebut hanya jika program tersebut berisi baris - baris perintah dengan sintaks yang benar. Kalau dalam program - program tersebut terdapat kesalahan sintaks maka proses akan berhenti dan menampilkan pesan - pesan kesalahan, yang kemudian dikenal sebagai Syntax errors. Sintaks merujuk ke sebuah struktur program dan aturan - aturan yang berperan dalam struktur tersebut. Sebagai contohnya, dalam bahasa Indonesia, sebuah kalimat harus di awali dengan huruf kapital dan dan diakhiri dengan tanda titik (.), kalimat tersebut akan mempunyai kesalahan sintaks jika penulisan kalimat tidak sesuai dengan aturan yang berlaku. Hal ini juga berlaku di dalam bahasa pemograman komputer.

Pada kebanyakan pembaca, beberapa kesalahan sintaks bukanlah masalah yang serius, seperti penulisan puisi, sajak dan lainnya. Tetapi bahasa pemograman Python bukanlah pemaaf yang baik dalam hal tersebut, jika terdapat satu kesalahan sintaks, maka program langsung memberikan pesan kesalahan dan keluar dari program. Pada waktu Anda baru mulai memprogram, mungkin Anda akan banyak menemui kesalahan -

kesalahan sintaks tersebut. Ketika Anda sudah terbiasa memprogram, Anda hanya akan menemui beberapa kesalahan dan menemukan kesalahan tersebut dengan cepat.

Kesalahan sintaks, dapat juga disebut dengan kesalahan dalam memparsing kode python yang salah, umumnya ditemui pada saat Anda baru memulai belajar bahasa pemograman python. Contohnya :

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

Pada contoh diatas, interpreter memberitahukan bahwa pada perintah terdapat kesalahan sintaks, interpreter akan menampilkan baris yang salah dan menunjukkan posisi kode yang salah dengan tanda panah kecil, contoh di atas pada penggunaan while seharusnya memberi tanda titik dua ":" setelah kondisi while.

2.1.2 Runtime errors

Jenis kesalahan yang kedua disebut dengan runtime errors, disebut begitu karena kesalahan tidak akan muncul sampai Anda menjalankan program tersebut. Kesalahan ini juga dikenal dengan exceptions atau pengecualian karena biasanya mengindikasikan sesuatu pengecualian yang buruk telah terjadi.
Runtime errors sangat jarang terjadi pada program - program yang sederhana.

2.1.3 Kesalahan Algoritma

Jenis kesalahan ketiga adalah kesalahan algoritma, yang kemudian dikenal dengan semantic errors. Jika terdapat kesalahan jenis ini dalam program Anda, program Anda akan berjalan dan tidak mengeluarkan pesan - pesan kesalahan, tetapi tidak akan sesuai dengan harapan Anda. Akan terjadi penyimpangan dari keinginan Anda.

Karena program tersebut tidak sesuai dengan harapan Anda dan akan meminta Anda untuk menelusuri kembali program tersebut dari awal untuk memperbaiki algoritmanya, kesalahan ini akan sering muncul pada saat Anda mulai berpengalaman dengan suatu bahasa pemograman.

2.1.4 Penulusuran kembali

Satu hal yang paling penting dan Anda harus punya adalah penulusuran kembali atau disebut kemudian sebagai debugging. Walaupun hal tersebut bisa membuat putus asa, debugging merupakan kekayaan intelektual seseorang yang paling tinggi, menantang dan bagian yang paling menarik dari pemograman.

Pada beberapa cara, debugging bekerja seperti halnya seorang detektif, Anda dipertemukan dengan berbagai petunjuk dan menulusuri sebuah proses dan kejadian - kejadian yang akhirnya mendapatkan hasil yang Anda inginkan.

Debugging juga seperti bereksperimen. Ketika Anda mendapatkan ide kesalahan apa yang telah terjadi, Anda memodifikasi dan mencobanya lagi. Jika hipotesis Anda

benar, Anda akan dapat menerka hasil dari modifikasi tersebut, dan selangkah lebih dekat dengan hasil akhir program tersebut.

Menurut pendapat beberapa orang, pemograman dan debugging adalah hal yang sama. Jadi pemograman adalah sebuah proses yang harus melalui proses beberapa kali debugging untuk mendapatkan hasil yang Anda inginkan. Pada contohnya, Linux adalah sebuah sistem operasi, yang berisikan beribu - ribu baris kode perintah, tetapi program tersebut diawali dengan program sederhana yang Linus Trovalds gunakan untuk mengeksplorasi chip Intel 80386. Berdasarkan keterangan Larry Greenfield, "Proyek pertama linus adalah sebuah program yang mencetak AAAA dan BBBB secara bergantian. Kemudian program ini berevolusi menjadi Linux." (Majalah Linux Users's Guide Beta Version 1)

2.1.5 Penulisan Komentar

Dalam proses debugging, suatu komentar instruksi program sangat berguna sekali dalam pembacaan suatu kode. Pada umumnya komentar berisi keterangan tentang kegunaan suatu fungsi itu. Sintaksnya adalah tanda kres atau tanda pagar "#". Setelah meletakkan tanda tersebut, kita dapat mengetikan kalimat apa saja yang berhubungan dengan suatu instruksi perintah, sebab apapun kalimat tersebut tidak akan di proses oleh interpreter. Contohnya :

```
print Hello World! #Mencetak string "Hello World!" ke layar.
print 4 + 5 #Menampilkan hasil dari bilangan 4 + 5.
```

Anda mungkin telah banyak melihat pesan - pesan kesalahan yang terjadi, terdapat dua jenis yang tidak dapat dipisahkan, yaitu : kesalahan sintaks dan pengecualian(exceptions).

2.2 Pengecualian (exceptions)

Jika terjadi kesalahan pada saat program dijalankan (run-time errors), program tersebut membuat sebuah pengecualian (exceptions). Biasanya program terhenti dan menampilkan pesan kesalahan. Contohnya pada pembagian bilangan dengan nol :

```
>>> 40 / 0
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

Juga untuk pengaksesan yang bukan elemen anggota dari suatu list.

```
>>> a = []
>>> a[5]
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
IndexError: list index out of range
```

Atau mengakses sebuah key yang tidak ada pada suatu dictionary.

```
>>> c = {}
>>> print c['polo']
Traceback (most recent call last):
      File "<stdin>", line 1, in ?
KeyError: polo
```

Pada setiap kasus, pesan kesalahan dibagi menjadi dua bagian: Jenis kesalahan sebelum titik dua, dan menjelaskan secara spesifik tentang kesalahan tersebut dibagian setelah titik dua. Pada umumnya, interpreter Python juga menampilkan sebuah penulusuran kembali dimana kesalahan pada program tersebut, yang dapat kita lihat sebelumnya.

2.3 Menangani pengecualian (Handling exceptions)

Kadang - kadang kita ingin menjalankan sebuah operasi yang dapat menyebabkan kesalahan pada runtime, tetapi kita tidak mau program tersebut berhenti total. Kita dapat mengatasinya dengan perintah try dan except.

Contohnya, Kita memberikan prompt pada user untuk menentukan nama file yang akan dibuka. Jika file tersebut tidak ada, kita tidak menginginkan program tersebut berhenti total yang dikarenakan tidak adanya file tersebut, kita menginginkan untuk mengendalikan kesalahan tersebut, dengan :

```
namafile = raw_input("Masukan nama file: ")
try :
      f = open (namafile, "r")
except :
      print "Nama file tidak ditemukan!"
```

Perintah try menjalankan perintah pada blok pertama. Jika tidak ada kesalahan yang terjadi, perintah except akan diabaikan. Tetapi jika terjadi kesalahan apapun jenisnya, program akan menjalankan perintah - perintah pada ruang lingkup except dan kemudian program tersebut berlanjut.

Kita dapat menyatukannya dan menjadikannya suatu fungsi, yaitu fungsi exist. fungsi ini akan mengambil argumen parameter sebuah nama file dan mengembalikan nilai true, jika file tersebut ada dan mengembalikan nilai false, jika file tersebut tidak ada. Contohnya :

```
def exist(namafile):
try :
      f = open (namafile)
      f.close()
      return 1
except :
      return 0
```

Anda dapat menggunakan kalimat perintah beberapa blok except (lebih dari satu) untuk menangani beberapa jenis kesalahan, detailnya Anda dapat lihat di Python Refference Manual. Jika pada program Anda mendeteksi adanya kesalahan, Anda dapat

membuatnya sebagai sebuah pengecualian (exception). Pada contoh ini kita akan membuat suatu fungsi yang mengecheck umur seseorang.

```
def check():
      umur = input("Masukan umur anda? ")
      if umur <= 17:
            raise 'DibawahUmur!', 'Anda harus 17 tahun keatas'
      return umur
check()
```

Maka program tersebut jika dijalankan dengan mengisi umur dibawah 17 tahun, akan menimbulkan suatu kesalahan exception.

```
Masukan umur anda? 3
Traceback (most recent call last):
      File "umur.py", line 6, in ?
      check()
File "umur.py", line 4, in check
raise 'DibawahUmur!', 'Anda harus 17 tahun keatas' DibawahUmur!: Anda
harus berumur 17 tahun keatas
```

Perintah raise akan mengambil dua argumen: jenis kesalahan dan informasi yang spesifik tentang kesalahan tersebut. DibawahUmur! adalah jenis kesalahan exception baru yang dibuat pada fungsi tersebut.

Latihan

Buatlah sebuah exception yang mengharuskan untuk memasukkan nilai minus (-1) dan menampilkan pesan kesalahan (exceptions) jika yang dimasukkan bukanlah tipe data bilangan integer atau float.

Jawaban

```
print "Ketikkan Control C atau -1 untuk keluar"
number = 1
while number != -1:
      try:
            number = int(raw_input("Masukan angka: "))
            print "Anda memasukan: ",number
      except ValueError:
            print "Bukan angka bilangan integer atau float!"
```