NAMA : MUHAMMAD FAIZ OKIRA
NIM :205410070

WORKSHOP (phython)

7. Masukan dan Keluaran

Ada beberapa cara untuk mempresentasikan output dari suatu program; data dapat dicetak dalam bentuk yang dapat dibaca manusia, atau ditulis ke file untuk digunakan di masa mendatang. Bab ini akan membahas beberapa kemungkinan.

7.1. Pemformatan Keluaran Lebih Menarik

Sejauh ini kita menemukan dua cara penulisan nilai: *pernyataan ekspresi* dan print()fungsi. (Cara ketiga menggunakan write()metode objek file; file keluaran standar dapat dirujuk sebagai sys.stdout. Lihat Referensi Perpustakaan untuk informasi lebih lanjut tentang ini.)

Seringkali Anda menginginkan kontrol lebih besar atas pemformatan keluaran Anda daripada sekadar mencetak nilai yang dipisahkan oleh ruang. Ada beberapa cara untuk memformat output.

- Untuk menggunakan literal string terformat , awali string dengan fatau Fsebelum tanda kutip pembuka atau tanda kutip tiga. Di dalam string ini, Anda dapat menulis ekspresi Python antara karakter {dan }yang dapat merujuk ke variabel atau nilai literal.

```
>>> year = 2016
>>> event ='Referendum'
>>> f'Results of the {year} {event}'
'Results of the 2016 Referendum'
```

- Metode str.format()string membutuhkan lebih banyak usaha manual. Anda masih akan menggunakan {dan }untuk menandai di mana variabel akan diganti dan dapat memberikan arahan pemformatan yang mendetail, tetapi Anda juga harus memberikan informasi yang akan diformat.

```
>>> yea_votes = 42_572_654
>>> yes_votes = 42_572_654
>>> no_votes =43_132_495
>>> percentage = yes_votes / (yes_votes)
>>> '{:-9} YES votes {:2.2%}'.format(yes_votes, percentage)
' 42572654 YES votes 100.00%'
```

- Terakhir, Anda dapat melakukan semua penanganan string sendiri dengan menggunakan pemotongan string dan operasi penggabungan untuk membuat tata letak apa pun yang dapat Anda bayangkan. Tipe string memiliki beberapa metode yang melakukan operasi berguna untuk mengisi string ke lebar kolom tertentu.

Saat Anda tidak membutuhkan output mewah tetapi hanya ingin tampilan cepat dari beberapa variabel untuk keperluan debugging, Anda dapat mengonversi nilai apa pun menjadi string dengan fungsi repr()atau str().

Fungsi ini str()dimaksudkan untuk mengembalikan representasi nilai yang dapat dibaca oleh manusia, sementara repr()dimaksudkan untuk menghasilkan representasi yang dapat dibaca oleh penafsir (atau akan memaksa a SyntaxErrorjika tidak ada sintaks yang setara). Untuk objek yang tidak memiliki representasi khusus untuk konsumsi manusia, str()akan mengembalikan nilai yang sama dengan repr(). Banyak nilai, seperti angka atau struktur seperti daftar dan kamus, memiliki representasi yang sama menggunakan salah satu fungsi. String, khususnya, memiliki dua representasi berbeda.

Beberapa contoh:

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr
(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
```

```
>>> hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> repr((x, y, ('spam','eggs')))
"(32.5, 40000, ('spam', 'eggs'))"
>>>
```

Modul stringberisi Templatekelas yang menawarkan cara lain untuk mengganti nilai menjadi string, menggunakan placeholder seperti $xdan menggantinya dengan nilai dari kamus, tetapi menawarkan kontrol pemformatan yang jauh lebih sedikit.

7.1.1. Literal String Terformat

Literal string terformat (disingkat juga disebut f-string) memungkinkan Anda menyertakan nilai ekspresi Python di dalam string dengan mengawali string dengan`forF`dan menulis ekspresi sebagai`{expression}`.

Penentu format opsional dapat mengikuti ekspresi. Ini memungkinkan kontrol yang lebih besar atas bagaimana nilai diformat. Contoh berikut membulatkan pi ke tiga tempat setelah desimal:

```
>>> import math
>>> print(f'The value of pi is approximately {math.pi:.3f}.')

The value of pi is approximately 3.142.
>>>
```

Melewati bilangan ':'bulat setelah akan menyebabkan bidang itu menjadi jumlah minimum karakter. Ini berguna untuk membuat kolom berbaris.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab' : 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
>>>
```

Pengubah lain dapat digunakan untuk mengonversi nilai sebelum diformat. '!a'berlaku ascii(), '!s'menerapkan str(), dan '!r'menerapkan repr():

```
>>> animals = 'eels'
>>> print(f'My hovercraft is full of {animals}.')
My hovercraft is full of eels.
>>> print(f'My hovercraft is full of {animals!r}.')
My hovercraft is full of 'eels'.
>>>
```

Penspesifikasi =dapat digunakan untuk memperluas ekspresi ke teks ekspresi, tanda sama dengan, lalu representasi ekspresi yang dievaluasi:

```
>>> bugs = 'roaches'
>>> count = 13
>>> area = 'living room'
>>> print(f' Debugging {bugs=} {count=} {area}')
 Debugging bugs='roaches' count=13 living room
>>>
```

Lihat ekspresi mendokumentasikan diri sendiri untuk informasi lebih lanjut tentang =specifier. Untuk referensi mengenai spesifikasi format ini, lihat panduan referensi untuk Spesifikasi Format Mini-Language .

7.1.2. Metode String format()

Penggunaan dasar metode ini str.format()terlihat seperti ini:

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni
'))
We are the knights who say "Ni!"
>>>
```

Tanda kurung dan karakter di dalamnya (disebut bidang format) diganti dengan objek yang diteruskan ke str.format()metode. Angka dalam tanda kurung dapat digunakan untuk merujuk ke posisi objek yang diteruskan ke metode str.format().

```
>>> print('This {food} is {adjective}.'.format(
...       food='spam', adjective= 'absolotely horrible'))
This spam is absolotely horrible.
>>>
```

Argumen posisi dan kata kunci dapat digabungkan secara sewenang-wenang:

```
>>> print('The story of {0}, {1}, and {other}.'.format('Bill'
, 'Manfred',
...                                                 other=
'Georg'))
The story of Bill, Manfred, and Georg.
>>>
```

Jika Anda memiliki format string yang sangat panjang yang tidak ingin Anda pisahkan, alangkah baiknya jika Anda dapat mereferensikan variabel untuk diformat dengan nama, bukan dengan posisi. Ini dapat dilakukan hanya dengan meneruskan dict dan menggunakan tanda kurung siku `'[]'`untuk mengakses kunci.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...       'Dcab: {0[Dcab]:d}'.format(table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
>>>
```

Ini juga bisa dilakukan dengan mengirimkan `table`kamus sebagai argumen kata kunci dengan `**`notasi.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dcab:d}
'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
>>>
```

Ini sangat berguna dalam kombinasi dengan fungsi bawaan vars(), yang mengembalikan kamus yang berisi semua variabel lokal.

Sebagai contoh, baris-baris berikut menghasilkan kumpulan kolom yang tertata rapi yang menghasilkan bilangan bulat dan kuadrat serta kubusnya:

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
>>>
```

Untuk ikhtisar lengkap tentang pemformatan string dengan str.format(), lihat Format Sintaks String .

7.1.3. Pemformatan String Manual

Berikut tabel kotak dan kubus yang sama, diformat secara manual:

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # Note use of 'end' on previous line
...     print(repr(x*x*x).rjust(4))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
>>>
```

(Perhatikan bahwa satu spasi di antara setiap kolom ditambahkan dengan cara print()kerjanya: selalu menambahkan spasi di antara argumennya.)

Metode str.rjust()objek string membenarkan string di bidang dengan lebar tertentu dengan melapisinya dengan spasi di sebelah kiri. Ada metode serupa str.ljust()dan str.center(). Metode ini tidak menulis apa pun, mereka hanya mengembalikan string baru. Jika string input terlalu panjang, mereka tidak memotongnya, tetapi mengembalikannya tanpa perubahan; ini akan mengacaukan tata letak kolom Anda tetapi itu biasanya lebih baik daripada alternatifnya, yang akan berbohong tentang suatu nilai. (Jika Anda benar-benar menginginkan pemotongan, Anda selalu dapat menambahkan operasi pemotongan, seperti pada `x.ljust(n)[:n]`.)

Ada metode lain, str.zfill(), yang mengisi string numerik di sebelah kiri dengan nol. Ia mengerti tentang tanda plus dan minus:

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
>>>
```

7.1.4. Pemformatan string lama

Operator % (modulo) juga dapat digunakan untuk pemformatan string. Diberikan , instance in diganti dengan nol atau lebih elemen . Operasi ini umumnya dikenal sebagai interpolasi string. Misalnya:`'string' % values%stringvalues`

```
>>> import math
>>> print('The value of pi is approximately %5.3f.' % math.pi
)
The value of pi is approximately 3.142.
>>>
```

Informasi lebih lanjut dapat ditemukan di bagian Pemformatan String gaya printf .

7.2. Membaca dan Menulis File

`open()`mengembalikan objek file , dan paling sering digunakan dengan dua argumen posisi dan satu argumen kata kunci:`open(filename, mode, encoding=None)`

```
>>> f = open('workfile', 'w', encoding="utf-8")
```

Argumen pertama adalah string yang berisi nama file. Argumen kedua adalah string lain yang berisi beberapa karakter yang menjelaskan cara penggunaan file. *mode* bisa `'r'`saat file hanya akan dibaca, `'w'`untuk ditulis saja (file yang sudah ada dengan nama yang sama akan dihapus), dan `'a'`membuka file untuk ditambahkan; data apa pun yang ditulis ke file secara otomatis ditambahkan ke akhir. `'r+'`membuka file untuk membaca dan menulis. Argumen *mode* bersifat opsional; `'r'`akan dianggap jika itu dihilangkan.

Biasanya, file dibuka dalam *mode teks* , artinya, Anda membaca dan menulis string dari dan ke file, yang dikodekan dalam *pengkodean* tertentu . Jika *penyandian* tidak ditentukan, defaultnya tergantung platform (lihat `open()`). Karena UTF-8 adalah standar de-facto modern, `encoding="utf-8"`disarankan kecuali Anda tahu bahwa Anda perlu menggunakan penyandian yang berbeda. Menambahkan a `'b'`ke mode akan membuka file dalam *mode biner* . Data mode biner dibaca dan ditulis sebagai `bytes`objek. Anda tidak dapat menentukan *penyandian* saat membuka file dalam mode biner.

Dalam mode teks, default saat membaca adalah mengonversi akhir baris khusus platform (`\n`di Unix, `\r\n`di Windows) menjadi hanya `\n`. Saat menulis dalam mode teks, defaultnya adalah mengonversi kejadian `\n`kembali ke akhir baris khusus platform. Modifikasi di balik layar untuk file data ini bagus untuk file teks, tetapi akan merusak data biner seperti itu di `JPEG`or `EXE`files. Berhati-hatilah untuk menggunakan mode biner saat membaca dan menulis file semacam itu.

Ini adalah praktik yang baik untuk menggunakan `with`kata kunci saat berhadapan dengan objek file. Keuntungannya adalah file ditutup dengan benar setelah rangkaiannya selesai, bahkan jika pengecualian dimunculkan di beberapa titik. Menggunakan `with`juga jauh lebih pendek daripada menulis yang setara `try`- `finally`blok:

```
>>> with open('workfile', encoding="utf-8") as f:
...     read_data = f.read()
...
>>> f.closed
True
>>>
```

Jika Anda tidak menggunakan withkata kunci, maka Anda harus memanggil f.close()untuk menutup file dan segera mengosongkan semua sumber daya sistem yang digunakan olehnya.

Peringatan

Memanggil f.write()tanpa menggunakan withkata kunci atau memanggil f.close() **dapat** mengakibatkan argumen f.write()tidak sepenuhnya ditulis ke disk, bahkan jika program berhasil keluar.

Setelah objek file ditutup, baik dengan withpernyataan atau dengan memanggil f.close(), upaya untuk menggunakan objek file akan gagal secara otomatis.

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

7.2.1. Metode File Objek

Contoh lainnya di bagian ini akan mengasumsikan bahwa objek file yang dipanggil ftelah dibuat.

Untuk membaca isi file, panggil f.read(size), yang membaca sejumlah data dan mengembalikannya sebagai string (dalam mode teks) atau objek byte (dalam mode biner). *size* adalah argumen numerik opsional. Ketika *ukuran* dihilangkan atau negatif, seluruh konten file akan dibaca dan dikembalikan; itu masalah Anda jika file tersebut dua kali lebih besar dari memori mesin Anda. Jika tidak, sebagian besar *ukuran* karakter (dalam mode teks) atau *ukuran* byte (dalam mode biner) dibaca dan dikembalikan. Jika akhir file telah tercapai, f.read()akan mengembalikan string kosong ('').

```
>>>
```

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

f.readline()membaca satu baris dari file; karakter baris baru (\n) dibiarkan di akhir string, dan hanya dihilangkan di baris terakhir file jika file tidak diakhiri di baris baru. Ini membuat nilai pengembalian tidak ambigu; jika f.readline()mengembalikan string kosong, akhir file telah tercapai, sedangkan baris kosong diwakili oleh '\n', string yang hanya berisi satu baris baru.

```
>>>
```

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Untuk membaca baris dari file, Anda dapat mengulang objek file. Ini hemat memori, cepat, dan mengarah ke kode sederhana:

```
>>>
```

```
>>> for line in f:
...     print(line, end='')
...
This is the first line of the file.
Second line of the file
```

Jika Anda ingin membaca semua baris file dalam daftar, Anda juga dapat menggunakan `list(f)`atau `f.readlines()`.

`f.write(string)`menulis konten *string* ke file, mengembalikan jumlah karakter yang ditulis.

```
>>>
```

```
>>> f.write('This is a test\n')
15
```

Jenis objek lain perlu dikonversi – baik menjadi string (dalam mode teks) atau objek byte (dalam mode biner) – sebelum menulisnya:

```
>>>
```

```
>>> value = ('the answer', 42)
>>> s = str(value)  # convert the tuple to string
>>> f.write(s)
18
```

`f.tell()`mengembalikan bilangan bulat yang memberikan posisi objek file saat ini dalam file yang direpresentasikan sebagai jumlah byte dari awal file saat dalam mode biner dan angka buram saat dalam mode teks.

Untuk mengubah posisi objek file, gunakan . Posisi dihitung dari penambahan *offset* ke titik referensi; titik referensi dipilih oleh argumen *wherece* . Nilai 0 mengukur dari *awal* file, 1 menggunakan posisi file saat ini, dan 2 menggunakan akhir file sebagai titik referensi. *dari mana* dapat dihilangkan dan default ke 0, menggunakan awal file sebagai titik referensi.`f.seek(offset, whence)`

```
>>>
```

```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)  # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

Dalam file teks (yang dibuka tanpa a `b`dalam string mode), hanya pencarian relatif ke awal file yang diperbolehkan (pengecualian sedang mencari hingga akhir file dengan) dan satu-satunya nilai *offset* yang valid adalah yang dikembalikan dari , atau nol. *Nilai offset* lainnya menghasilkan perilaku yang tidak terdefinisi.`seek(0, 2)`f.tell()

Objek file memiliki beberapa metode tambahan, seperti `isatty()`dan `truncate()`yang jarang digunakan; berkonsultasi dengan Referensi Perpustakaan untuk panduan lengkap untuk objek file.

7.2.2. Menyimpan data terstruktur dengan json

String dapat dengan mudah ditulis dan dibaca dari file. Angka membutuhkan sedikit lebih banyak usaha, karena `read()`metode ini hanya mengembalikan string, yang harus diteruskan ke fungsi seperti `int()`, yang mengambil string seperti `'123'`dan mengembalikan nilai numeriknya 123. Saat Anda ingin menyimpan tipe data yang lebih kompleks seperti daftar bersarang dan kamus, parsing dan serialisasi dengan tangan menjadi rumit.

Daripada membuat pengguna terus-menerus menulis dan men-debug kode untuk menyimpan tipe data yang rumit ke file, Python memungkinkan Anda untuk menggunakan format pertukaran data populer yang disebut JSON (JavaScript Object Notation) . Modul standar yang dipanggil `json`dapat mengambil hierarki data Python, dan mengubahnya menjadi representasi string; proses ini disebut *serialisasi* . Merekonstruksi data dari representasi string disebut *deserializing* . Antara serialisasi dan deserialisasi, string yang mewakili objek mungkin telah disimpan dalam file atau data, atau dikirim melalui koneksi jaringan ke beberapa mesin yang jauh.

Catatan

Format JSON umumnya digunakan oleh aplikasi modern untuk memungkinkan pertukaran data. Banyak pemrogram sudah mengenalnya, yang menjadikannya pilihan yang baik untuk interoperabilitas.

Jika Anda memiliki objek `x`, Anda dapat melihat representasi string JSON dengan baris kode sederhana:

```
>>> import json
>>> x = [1, 'simple', 'list']
>>> json.dumps(x)
'[1, "simple", "list"]'
>>>
```

Varian lain dari dumps()fungsi, disebut dump(), cukup membuat serial objek menjadi file teks . Jadi jika fobjek file teks dibuka untuk menulis, kita dapat melakukan ini:

```
>>> x = json.load(f)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\M415\anaconda3\lib\json\__init__.py", line 2
93, in load
    return loads(fp.read(),
ValueError: I/O operation on closed file.
>>>
```

Untuk mendekode ulang objek, jika ffile biner atau objek file teks yang telah dibuka untuk dibaca:

```
x = json.load(f)
```

Catatan

File JSON harus dikodekan dalam UTF-8. Gunakan encoding="utf-8"saat membuka file JSON sebagai file teks untuk membaca dan menulis.

Teknik serialisasi sederhana ini dapat menangani daftar dan kamus, tetapi membuat serialisasi instance kelas arbitrer di JSON membutuhkan sedikit usaha ekstra. Referensi untuk jsonmodul berisi penjelasan tentang hal ini.