Struktur data

Struktur data pada dasarnya hanya itu - mereka adalah *struktur* yang dapat menampung beberapa *data* bersama-sama. Dengan kata lain, mereka digunakan untuk menyimpan kumpulan data terkait.

Ada empat struktur data bawaan dalam Python - *list, tuple, dictionary dan set* . Kita akan melihat bagaimana menggunakan masing-masing dari mereka dan bagaimana mereka membuat hidup kita lebih mudah.

Daftar

A `list` adalah struktur data yang menyimpan koleksi item yang dipesan yaitu Anda dapat menyimpan *urutan* item dalam daftar. Ini mudah untuk dibayangkan jika Anda dapat memikirkan daftar belanja di mana Anda memiliki daftar barang untuk dibeli, kecuali bahwa Anda mungkin memiliki setiap item pada baris terpisah dalam daftar belanja Anda sedangkan dengan Python Anda meletakkan koma di antara mereka.

Daftar item harus diapit dalam tanda kurung siku sehingga Python mengerti bahwa Anda sedang menentukan daftar. Setelah Anda membuat daftar, Anda dapat menambahkan, menghapus, atau mencari item dalam daftar. Karena kita dapat menambah dan menghapus item, kita mengatakan bahwa daftar adalah *tipe* data yang dapat diubah yaitu tipe ini dapat diubah.

Pengantar Cepat Untuk Objek Dan Kelas

Meskipun saya biasanya menunda diskusi tentang objek dan kelas sampai sekarang, sedikit penjelasan diperlukan sekarang sehingga Anda dapat memahami daftar dengan lebih baik. Kami akan mengeksplorasi topik ini secara rinci dalam bab selanjutnya .

Daftar adalah contoh penggunaan objek dan kelas. Ketika kita menggunakan variabel `i` dan memberikan nilai padanya, katakanlah integer `5` padanya, Anda dapat menganggapnya sebagai membuat *objek* (yaitu instance) `i` dari *class* (yaitu type) `int` . Bahkan, Anda dapat membaca `help(int)` untuk memahami ini lebih baik.

Sebuah kelas juga dapat memiliki *metode* yaitu fungsi yang didefinisikan untuk digunakan sehubungan dengan kelas itu saja. Anda dapat menggunakan bagian fungsionalitas ini hanya jika Anda memiliki objek dari kelas tersebut. Misalnya, Python menyediakan `append` metode untuk `list` kelas yang memungkinkan Anda menambahkan item ke akhir daftar. Misalnya, `mylist.append('an item')` akan menambahkan string itu ke daftar `mylist` . Perhatikan penggunaan notasi titik-titik untuk mengakses metode objek.

Kelas juga dapat memiliki *bidang* yang tidak lain adalah variabel yang ditentukan untuk digunakan sehubungan dengan kelas itu saja. Anda dapat menggunakan variabel/nama ini hanya jika Anda memiliki objek dari kelas tersebut. Bidang juga diakses oleh notasi putus-putus, misalnya, `mylist.field` .

Contoh (simpan sebagai `ds_using_list.py`):

```
# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print('I have', len(shoplist), 'items to purchase.')

print('These items are:', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('My shopping list is now', shoplist)

print('I will sort my list now')
shoplist.sort()
print('Sorted shopping list is', shoplist)

print('The first item I will buy is', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('I bought the', olditem)
print('My shopping list is now', shoplist)
```

Keluaran:

```
$ python ds_using_list.py
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

Bagaimana itu bekerja

Variabelnya `shoplist` adalah daftar belanja seseorang yang akan pergi ke pasar. Di `shoplist` , kami hanya menyimpan string nama item yang akan dibeli tetapi Anda dapat menambahkan *objek apa pun* ke daftar termasuk angka dan bahkan daftar lainnya.

Kami juga telah menggunakan `for..in` loop untuk beralih melalui item dari daftar. Sekarang, Anda pasti sudah menyadari bahwa daftar juga merupakan urutan. Keistimewaan urutan akan dibahas di bagian selanjutnya .

Perhatikan penggunaan `end` parameter dalam panggilan ke `print` fungsi untuk menunjukkan bahwa kita ingin mengakhiri output dengan spasi alih-alih jeda baris biasa.

Selanjutnya, kita menambahkan item ke daftar menggunakan `append` metode objek daftar, seperti yang telah dibahas sebelumnya. Kemudian, kami memeriksa apakah item tersebut memang telah ditambahkan ke daftar dengan mencetak isi daftar hanya dengan meneruskan daftar ke `print` fungsi yang mencetaknya dengan rapi.

Kemudian, kami mengurutkan daftar dengan menggunakan `sort` metode daftar. Penting untuk dipahami bahwa metode ini memengaruhi daftar itu sendiri dan tidak mengembalikan daftar yang dimodifikasi - ini berbeda dari cara kerja string. Inilah yang kami maksud dengan mengatakan bahwa daftar dapat *diubah* dan bahwa string tidak *dapat diubah* .

Selanjutnya, ketika kami selesai membeli suatu barang di pasar, kami ingin menghapusnya dari daftar. Kami mencapai ini dengan menggunakan `del` pernyataan. Di sini, kami menyebutkan item mana dari daftar yang ingin kami hapus dan `del` pernyataan tersebut menghapusnya dari daftar untuk kami. Kami menentukan bahwa kami ingin menghapus item pertama dari daftar dan karenanya kami menggunakan `del shoplist[0]` (ingat bahwa Python mulai menghitung dari 0).

Jika Anda ingin mengetahui semua metode yang ditentukan oleh objek daftar, lihat `help(list)` detailnya.

Tupel

Tuple digunakan untuk menyatukan beberapa objek. Anggap mereka mirip dengan daftar, tetapi tanpa fungsionalitas ekstensif yang diberikan kelas daftar kepada Anda. Salah satu fitur utama dari tupel adalah bahwa mereka tidak *dapat diubah* seperti string yaitu Anda tidak dapat memodifikasi tupel.

Tuple didefinisikan dengan menentukan item yang dipisahkan oleh koma dalam sepasang kurung opsional.

Tuple biasanya digunakan dalam kasus di mana pernyataan atau fungsi yang ditentukan pengguna dapat dengan aman mengasumsikan bahwa kumpulan nilai (yaitu tupel nilai yang digunakan) tidak akan berubah.

Contoh (simpan sebagai `ds_using_tuple.py`):

```
# I would recommend always using parentheses
# to indicate start and end of tuple

# even though parentheses are optional.
# Explicit is better than implicit.
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = 'monkey', 'camel', zoo    # parentheses not required but are a good ide
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
      len(new_zoo)-1+len(new_zoo[2]))
```

Keluaran:

```
$ python ds_using_tuple.py
Number of animals in the zoo is 3
Number of cages in the new zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python', 'elephant', 'penguin'))
Animals brought from old zoo are ('python', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
Number of animals in the new zoo is 5
```

Bagaimana itu bekerja

Variabel `zoo` mengacu pada tuple item. Kita melihat bahwa `len` fungsi tersebut dapat digunakan untuk mendapatkan panjang tupel. Ini juga menunjukkan bahwa tupel adalah urutan juga.

Kami sekarang memindahkan hewan-hewan ini ke kebun binatang baru karena kebun binatang lama ditutup. Oleh karena itu, `new_zoo` tupel berisi beberapa hewan yang sudah ada bersama dengan hewan yang dibawa dari kebun binatang tua. Kembali ke kenyataan, perhatikan bahwa tupel di dalam tupel tidak kehilangan identitasnya.

Kita dapat mengakses item dalam tupel dengan menentukan posisi item dalam sepasang tanda kurung siku seperti yang kita lakukan untuk daftar. Ini disebut operator *pengindeksan .* Kami mengakses item ketiga `new_zoo` dengan menentukan `new_zoo[2]` dan kami mengakses item ketiga dalam item ketiga di `new_zoo` tuple dengan menentukan `new_zoo[2][2]` . Ini cukup sederhana setelah Anda memahami idiomnya.

> **Tuple dengan 0 atau 1 item**
>
> Tuple kosong dibangun oleh sepasang tanda kurung kosong seperti `myempty = ()` . Namun, tuple dengan satu item tidak sesederhana itu. Anda harus menentukannya menggunakan koma setelah item pertama (dan satu-satunya) sehingga Python dapat membedakan antara Tuple dan sepasang tanda kurung yang mengelilingi objek dalam ekspresi yaitu Anda harus menentukan `singleton = (2 , )` apakah yang Anda maksud adalah Tuple yang berisi item `2` .

> **Catatan untuk programmer Perl**
>
> Daftar di dalam daftar tidak kehilangan identitasnya yaitu daftar tidak diratakan seperti di Perl. Hal yang sama berlaku untuk tupel di dalam tupel, atau tupel di dalam daftar, atau daftar di dalam tupel, dll. Sejauh menyangkut Python, mereka hanyalah objek yang disimpan menggunakan objek lain, itu saja.

Kamus

Kamus seperti buku alamat di mana Anda dapat menemukan alamat atau detail kontak seseorang dengan hanya mengetahui namanya, yaitu kami mengaitkan *kunci* (nama) dengan *nilai* (detail). Perhatikan bahwa kunci harus unik seperti Anda tidak dapat menemukan informasi yang benar jika Anda memiliki dua orang dengan nama yang sama persis.

Perhatikan bahwa Anda hanya dapat menggunakan objek yang tidak dapat diubah (seperti string) untuk kunci kamus tetapi Anda dapat menggunakan objek yang tidak dapat diubah atau yang dapat diubah untuk nilai kamus. Ini pada dasarnya diterjemahkan untuk mengatakan bahwa Anda harus menggunakan hanya objek sederhana untuk kunci.

Pasangan kunci dan nilai ditentukan dalam kamus dengan menggunakan notasi `d = {key1 : value1, key2 : value2 }` . Perhatikan bahwa pasangan nilai kunci dipisahkan oleh titik dua dan pasangan dipisahkan oleh koma dan semua ini diapit oleh sepasang kurung kurawal.

Ingatlah bahwa pasangan nilai kunci dalam kamus tidak diurutkan dengan cara apa pun. Jika Anda menginginkan pesanan tertentu, maka Anda harus menyortirnya sendiri sebelum menggunakannya.

Kamus yang akan Anda gunakan adalah instance/objek dari `dict` kelas.

Contoh (simpan sebagai `ds_using_dict.py`):

```
# 'ab' is short for 'a'ddress'b'ook

ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}

print("Swaroop's address is", ab['Swaroop'])

# Deleting a key-value pair
del ab['Spammer']

print('\nThere are {} contacts in the address-book\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nGuido's address is", ab['Guido'])
```

Keluaran:

```
$ python ds_using_dict.py
Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org

Guido's address is guido@python.org
```

Bagaimana itu bekerja

Kami membuat kamus `ab` menggunakan notasi yang sudah dibahas. Kami kemudian mengakses pasangan nilai kunci dengan menentukan kunci menggunakan operator pengindeksan seperti yang dibahas dalam konteks daftar dan tupel. Perhatikan sintaks sederhana.

Kita dapat menghapus pasangan nilai kunci menggunakan teman lama kita - `del` pernyataan. Kami hanya menentukan kamus dan operator pengindeksan untuk kunci yang akan dihapus dan meneruskannya ke `del` pernyataan. Tidak perlu mengetahui nilai yang sesuai dengan kunci untuk operasi ini.

Selanjutnya, kita mengakses setiap pasangan kunci-nilai kamus menggunakan `items` metode kamus yang mengembalikan daftar tupel di mana setiap tupel berisi sepasang item - kunci diikuti oleh nilainya. Kami mengambil pasangan ini dan menetapkannya ke variabel `name` dan `address` secara bersamaan untuk setiap pasangan menggunakan `for..in` loop dan kemudian mencetak nilai-nilai ini di blok-for.

Kita dapat menambahkan pasangan nilai kunci baru hanya dengan menggunakan operator pengindeksan untuk mengakses kunci dan menetapkan nilai itu, seperti yang telah kita lakukan untuk Guido dalam kasus di atas.

Kita dapat memeriksa apakah ada pasangan nilai kunci menggunakan `in` operator.

Untuk daftar metode `dict` kelas, lihat `help(dict)` .

> **Argumen dan Kamus Kata Kunci**
>
> Jika Anda telah menggunakan argumen kata kunci dalam fungsi Anda, Anda telah menggunakan kamus! Pikirkan saja - pasangan nilai kunci ditentukan oleh Anda dalam daftar parameter definisi fungsi dan ketika Anda mengakses variabel dalam fungsi Anda, itu hanya akses kunci kamus (yang disebut *tabel simbol* dalam desain kompiler terminologi).

Urutan

Daftar, tupel, dan string adalah contoh urutan, tetapi apa itu urutan dan apa istimewanya?

Fitur utama adalah *tes keanggotaan* , (yaitu ekspresi dan) dan `in` operasi *pengindeksan* , yang memungkinkan kita untuk mengambil item tertentu dalam urutan secara langsung. `not in`

Tiga jenis urutan yang disebutkan di atas - daftar, tupel dan string, juga memiliki operasi *slicing* yang memungkinkan kita untuk mengambil sepotong urutan yaitu bagian dari urutan.

Contoh (simpan sebagai `ds_seq.py`):

```
shoplist = ['apple', 'mango', 'carrot', 'banana']
name = 'swaroop'

# Indexing or 'Subscription' operation #
print('Item 0 is', shoplist[0])
print('Item 1 is', shoplist[1])
print('Item 2 is', shoplist[2])
print('Item 3 is', shoplist[3])
print('Item -1 is', shoplist[-1])
print('Item -2 is', shoplist[-2])
print('Character 0 is', name[0])

# Slicing on a list #
print('Item 1 to 3 is', shoplist[1:3])
print('Item 2 to end is', shoplist[2:])
print('Item 1 to -1 is', shoplist[1:-1])
print('Item start to end is', shoplist[:])

# Slicing on a string #
print('characters 1 to 3 is', name[1:3])
print('characters 2 to end is', name[2:])
print('characters 1 to -1 is', name[1:-1])
print('characters start to end is', name[:])
```

Keluaran:

```
$ python ds_seq.py
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Character 0 is s
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

Bagaimana itu bekerja

Pertama, kita melihat bagaimana menggunakan indeks untuk mendapatkan item individual dari suatu urutan. Ini juga disebut sebagai *operasi berlangganan* . Setiap kali Anda menentukan nomor ke urutan dalam tanda kurung siku seperti yang ditunjukkan di atas, Python akan mengambilkan Anda item yang sesuai dengan posisi itu dalam urutan. Ingat bahwa Python mulai menghitung angka dari 0. Oleh karena itu, `shoplist[0]` mengambil item pertama dan `shoplist[3]` mengambil item keempat dalam `shoplist` urutan.

Indeks juga bisa berupa angka negatif, dalam hal ini, posisi dihitung dari akhir urutan. Oleh karena itu, `shoplist[-1]` mengacu pada item terakhir dalam urutan dan `shoplist[-2]` mengambil item terakhir kedua dalam urutan.

Operasi slicing digunakan dengan menentukan nama urutan diikuti oleh pasangan opsional angka yang dipisahkan oleh titik dua di dalam tanda kurung siku. Perhatikan bahwa ini sangat mirip dengan operasi pengindeksan yang telah Anda gunakan sampai sekarang. Ingat angkanya opsional tetapi titik dua tidak.

Angka pertama (sebelum titik dua) dalam operasi pengirisan mengacu pada posisi dari mana potongan dimulai dan angka kedua (setelah titik dua) menunjukkan di mana potongan akan berhenti. Jika nomor pertama tidak ditentukan, Python akan mulai dari awal urutan. Jika

nomor kedua ditinggalkan, Python akan berhenti di akhir urutan. Perhatikan bahwa irisan yang dikembalikan *dimulai* pada posisi awal dan akan berakhir tepat sebelum posisi *akhir* yaitu posisi awal disertakan tetapi posisi akhir dikeluarkan dari irisan urutan.

Jadi, `shoplist[1:3]` mengembalikan sepotong urutan yang dimulai dari posisi 1, termasuk posisi 2 tetapi berhenti di posisi 3 dan oleh karena itu *potongan* dua item dikembalikan. Demikian pula, `shoplist[:]` mengembalikan salinan seluruh urutan.

Anda juga bisa melakukan slicing dengan posisi negatif. Angka negatif digunakan untuk posisi dari akhir barisan. Misalnya, `shoplist[:-1]` akan mengembalikan sepotong urutan yang mengecualikan item terakhir dari urutan tetapi berisi yang lainnya.

Anda juga dapat memberikan argumen ketiga untuk irisan, yang merupakan *langkah* untuk irisan (secara default, ukuran langkah adalah 1):

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::1]
['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::2]
['apple', 'carrot']
>>> shoplist[::3]
['apple', 'banana']
>>> shoplist[::-1]
['banana', 'carrot', 'mango', 'apple']
```

Perhatikan bahwa ketika langkahnya adalah 2, kita mendapatkan item dengan posisi 0, 2,... Ketika ukuran langkahnya adalah 3, kita mendapatkan item dengan posisi 0, 3, dst.

Cobalah berbagai kombinasi spesifikasi irisan tersebut menggunakan interpreter Python secara interaktif yaitu prompt sehingga Anda dapat melihat hasilnya dengan segera. Hal terbaik tentang sequence adalah Anda dapat mengakses tuple, list, dan string dengan cara yang sama!

Mengatur

Himpunan adalah kumpulan objek sederhana yang *tidak berurutan* . Ini digunakan ketika keberadaan objek dalam koleksi lebih penting daripada urutan atau berapa kali itu terjadi.

Dengan menggunakan himpunan, Anda dapat menguji keanggotaan, apakah itu himpunan bagian dari himpunan lain, menemukan perpotongan antara dua himpunan, dan seterusnya.

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Bagaimana itu bekerja

Jika Anda ingat matematika teori himpunan dasar dari sekolah, maka contoh ini cukup jelas. Tetapi jika tidak, Anda dapat google "teori himpunan" dan "diagram Venn" untuk lebih memahami penggunaan himpunan kami dengan Python.

Referensi

Saat Anda membuat objek dan menetapkannya ke variabel, variabel hanya *merujuk* ke objek dan tidak mewakili objek itu sendiri! Artinya, nama variabel menunjuk ke bagian memori komputer tempat objek disimpan. Ini disebut *mengikat* nama ke objek.

Secara umum, Anda tidak perlu khawatir tentang ini, tetapi ada efek halus karena referensi yang perlu Anda perhatikan:

Contoh (simpan sebagai `ds_reference.py`):

```
print('Simple Assignment')
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist

# I purchased the first item, so I remove it from the list
del shoplist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that both shoplist and mylist both print
# the same list without the 'apple' confirming that
# they point to the same object

print('Copy by making a full slice')
# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that now the two lists are different
```

Keluaran:

```
$ python ds_reference.py
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

Bagaimana itu bekerja

Sebagian besar penjelasan tersedia di komentar.

Ingat bahwa jika Anda ingin membuat salinan dari daftar atau semacam urutan atau objek kompleks (bukan *objek* sederhana seperti bilangan bulat), maka Anda harus menggunakan operasi slicing untuk membuat salinan. Jika Anda hanya menetapkan nama variabel ke nama lain, keduanya akan "mengacu" ke objek yang sama dan ini bisa menjadi masalah jika Anda tidak hati-hati.

> **Catatan untuk programmer Perl**
>
> Ingat bahwa pernyataan tugas untuk daftar **tidak** membuat salinan. Anda harus menggunakan operasi pemotongan untuk membuat salinan urutan.

Lebih Lanjut Tentang String

Kami telah membahas string secara rinci sebelumnya. Apa lagi yang bisa diketahui? Nah, tahukah Anda bahwa string juga merupakan objek dan memiliki metode yang melakukan segalanya mulai dari memeriksa bagian string hingga menghilangkan spasi? Bahkan, Anda telah menggunakan metode string... `format` metode!

String yang Anda gunakan dalam program adalah semua objek dari kelas `str` . Beberapa metode yang berguna dari kelas ini ditunjukkan dalam contoh berikut. Untuk daftar lengkap metode tersebut, lihat `help(str)` .

Contoh (simpan sebagai `ds_str_methods.py`):

```
# This is a string object
name = 'Swaroop'

if name.startswith('Swa'):
    print('Yes, the string starts with "Swa"')

if 'a' in name:
    print('Yes, it contains the string "a"')

if name.find('war') != -1:
    print('Yes, it contains the string "war"')

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(mylist))
```

Keluaran:

```
$ python ds_str_methods.py
Yes, the string starts with "Swa"
Yes, it contains the string "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```

Bagaimana itu bekerja

Di sini, kita melihat banyak metode string beraksi. Metode `startswith` ini digunakan untuk mengetahui apakah string dimulai dengan string yang diberikan. Operator `in` digunakan untuk memeriksa apakah string yang diberikan adalah bagian dari string.

Metode `find` ini digunakan untuk menemukan posisi substring yang diberikan dalam string; `find` mengembalikan -1 jika tidak berhasil menemukan substring. Kelas `str` juga memiliki metode rapi untuk `join` item urutan dengan string yang bertindak sebagai pembatas antara setiap item dari urutan dan mengembalikan string yang lebih besar yang dihasilkan dari ini.

Ringkasan

Kami telah menjelajahi berbagai struktur data bawaan Python secara rinci. Struktur data ini akan sangat penting untuk menulis program dengan ukuran yang wajar.

Sekarang setelah kita memiliki banyak dasar-dasar Python, selanjutnya kita akan melihat bagaimana merancang dan menulis program Python dunia nyata.