Python for Everybody

Exploring Data Using Python 3

Dr. Charles R. Severance

Credits

Editorial Support: Elliott Hauser, Sue Blumenberg
Cover Design: Aimee Andrion

Printing History

- 2024-Jan-01 Update examples to Python 3.12, remove references to Twitter APIs, rewrite Databases chapter
- 2023-Jun-29 Many errata included, switch from Google APIs to OpenStreetMap APIs
- 2016-Jul-05 First Complete Python 3.0 version
- 2015-Dec-20 Initial Python 3.0 rough conversion

Copyright Details

Paunang Salita

Remixing an Open Book

Natural lang para sa mga academics na palaging sinasabihan na "publish or perish" na gusto nilang palaging gumawa ng bagong bagay mula sa simula na kanilang sariling original na gawa. Ang librong ito ay isang experiment na hindi nagsisimula sa simula, kundi "remixing" ang libro na may pamagat na *Think Python: How to Think Like a Computer Scientist* na sinulat ni Allen B. Downey, Jeff Elkner, at iba pa.

Noong December 2009, naghahanda ako na magturo ng *SI502 - Networked Programming* sa University of Michigan sa ikalimang semester na sunud-sunod at nagdesisyon na panahon na para sumulat ng Python textbook na nakatuon sa pag-explore ng data imbes na pag-intindi ng algorithms at abstractions. Ang goal ko sa SI502 ay turuan ang mga tao ng lifelong data handling skills gamit ang Python. Kakaunti lang sa mga estudyante ko ang nagpaplano na maging professional computer programmers. Sa halip, plano nilang maging librarians, managers, lawyers, biologists, economists, etc., na gusto lang na magaling na gumamit ng technology sa kanilang napiling field.

Parang hindi ko mahanap ang perfect na data-oriented Python book para sa course ko, kaya nagdesisyon akong sumulat ng ganitong libro. Swerte na lang sa faculty meeting tatlong linggo bago ako magsimula ng bagong libro mula sa simula sa holiday break, ipinakita sa akin ni Dr. Atul Prakash ang *Think Python* book na ginamit niya sa pagturo ng Python course niya noong semester na iyon. Ito ay isang well-written Computer Science text na nakatuon sa maikli, direkta na explanations at madaling matutunan.

Ang overall na book structure ay binago para makarating agad sa paggawa ng data analysis problems at may series ng running examples at exercises tungkol sa data analysis mula pa sa simula.

Ang Chapters 2–10 ay katulad ng *Think Python* book, pero may major changes. Ang number-oriented examples at exercises ay pinalitan ng data-oriented exercises. Ang topics ay inilatag sa order na kailangan para makabuo ng mas sophisticated na data analysis solutions. Ang ilang topics tulad ng `try` at `except` ay inilagay sa harap at ipinakita bilang parte ng chapter sa conditionals. Ang Functions ay binigyan ng very light treatment hanggang kailangan na sila para sa program complexity imbes na ipakita bilang early lesson sa abstraction. Halos lahat ng user-defined functions ay tinanggal sa example code at exercises maliban sa Chapter 4. Ang salitang "recursion"[1] ay hindi lumabas sa libro.

Sa chapters 1 at 11–16, lahat ng material ay brand new, nakatuon sa real-world uses at simple examples ng Python para sa data analysis kasama ang regular expressions para sa searching at parsing, automating tasks sa computer mo, retrieving data sa network, scraping web pages para sa data, object-oriented programming, pag-gamit ng web services, parsing XML at JSON data, pag-gawa at pag-gamit ng databases gamit ang Structured Query Language, at visualizing data.

Ang ultimate goal ng lahat ng changes na ito ay lumipat mula sa Computer Science patungo sa Informatics focus at isama lang ang topics sa first technology class na

[1] Except, of course, for this line.

useful kahit hindi mo pipiliin na maging professional programmer.

Ang mga estudyante na interesado sa librong ito at gusto pang mag-explore dapat tingnan ang *Think Python* book ni Allen B. Downey. Dahil may maraming overlap sa pagitan ng dalawang libro, mabilis na matututunan ng mga estudyante ang skills sa additional areas ng technical programming at algorithmic thinking na sakop sa *Think Python*. At dahil pareho ang writing style ng mga libro, dapat makakagalaw sila nang mabilis sa *Think Python* nang minimal na effort.

Nararamdaman ko na ang librong ito ay nagsisilbing halimbawa kung bakit ang open materials ay napaka important sa future ng education, at gusto kong magpasalamat kay Allen B. Downey at Cambridge University Press para sa kanilang forward-looking decision na gawing available ang libro sa ilalim ng open copyright. Sana masaya sila sa results ng efforts ko at sana ikaw, ang reader, ay masaya sa *aming* collective efforts.

Gusto kong magpasalamat kay Allen B. Downey at Lauren Cowles para sa kanilang tulong, patience, at guidance sa pag-deal at pag-resolve ng copyright issues tungkol sa librong ito.

Charles Severance
www.dr-chuck.com
Ann Arbor, MI, USA
September 9, 2013

Si Charles Severance ay isang Clinical Associate Professor sa University of Michigan School of Information.

Contents

Chapter 1

Bakit dapat mong matutunan ang pagsusulat ng programs?

Ang pagsusulat ng programs (o programming) ay isang napaka-creative at rewarding na activity. Maaari kang sumulat ng programs para sa maraming dahilan, mula sa paggawa ng ikabubuhay mo hanggang sa pag-solve ng mahirap na data analysis problem hanggang sa pag-enjoy hanggang sa pagtulong sa iba na mag-solve ng problema. Ang librong ito ay nag-aassume na *everyone* ay kailangang malaman kung paano mag-program, at kapag alam mo na kung paano mag-program ay malalaman mo kung ano ang gusto mong gawin sa iyong bagong natutunang skills.

Napapaligiran tayo sa ating daily lives ng mga computers mula sa laptops hanggang sa cell phones. Maaari nating isipin ang mga computers na ito bilang ating "personal assistants" na maaaring mag-alaga ng maraming bagay para sa atin. Ang hardware sa ating current-day computers ay essentially built para patuloy na magtanong sa atin ng tanong, "Ano ang gusto mong gawin ko next?"

Ang mga programmers ay nagdadagdag ng operating system at isang set ng applications sa hardware at napupunta tayo sa Personal Digital Assistant na napaka helpful at capable na tumulong sa atin na gumawa ng maraming iba't ibang bagay.

Ang ating mga computers ay mabilis at may malawak na memory at maaaring maging napaka helpful sa atin kung alam lang natin ang language na sasabihin

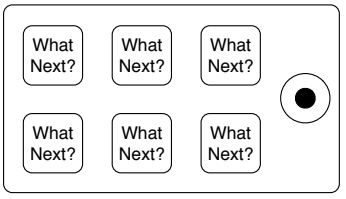

Figure 1.1: Personal Digital Assistant

para ipaliwanag sa computer kung ano ang gusto nating "gawin next". Kung alam natin ang language na ito, maaari nating sabihin sa computer na gawin ang mga tasks para sa atin na repetitive. Kapansin-pansin, ang mga bagay na kaya ng computers na gawin ay madalas ang mga bagay na nakakabagot at mind-numbing para sa atin na mga tao.

Halimbawa, tingnan ang unang tatlong paragraphs ng chapter na ito at sabihin mo sa akin ang pinaka-commonly used na salita at ilang beses ginamit ang salita. Habang nakaya mong basahin at maintindihan ang mga salita sa ilang segundo, ang pagbilang sa kanila ay halos masakit dahil hindi ito ang uri ng problema na idinisenyo ng human minds na solusyonan. Para sa computer, ang kabaligtaran ang totoo, ang pagbasa at pag-intindi ng text mula sa papel ay mahirap para sa computer na gawin pero ang pagbilang ng mga salita at pagsasabi sa iyo kung ilang beses ang pinaka-ginamit na salita ay napakadali para sa computer:

```
python words.py
Ilagay ang file: words.txt
to 16
```

Ang ating "personal information analysis assistant" ay mabilis na nagsabi sa atin na ang salitang "to" ay ginamit ng labing-anim na beses sa unang tatlong paragraphs ng chapter na ito.

Ang katotohanang ito na ang computers ay magaling sa mga bagay na hindi kaya ng mga tao ay dahilan kung bakit kailangan mong maging skilled sa pagsasalita ng "computer language". Kapag natutunan mo ang bagong language na ito, maaari mong idelegate ang mundane tasks sa iyong partner (ang computer), na nag-iiwan ng mas maraming oras para sa iyo na gawin ang mga bagay na ikaw lang ang uniquely suited para gawin. Nagdadala ka ng creativity, intuition, at inventiveness sa partnership na ito.

1.1 Creativity and motivation

Habang ang librong ito ay hindi intended para sa professional programmers, ang professional programming ay maaaring maging napaka-rewarding na trabaho pareho sa financially at personally. Ang paggawa ng useful, elegant, at clever programs para sa iba na gamitin ay isang napaka-creative na activity. Ang iyong computer o Personal Digital Assistant (PDA) ay karaniwang naglalaman ng maraming iba't ibang programs mula sa maraming iba't ibang grupo ng programmers, bawat isa ay nakikipag-compete para sa iyong attention at interest. Sinusubukan nila ang kanilang makakaya para matugunan ang iyong pangangailangan at bigyan ka ng great user experience sa proseso. Sa ilang situations, kapag pumili ka ng piece ng software, ang mga programmers ay direktang compensated dahil sa iyong choice.

Kung iisipin natin ang programs bilang creative output ng mga grupo ng programmers, marahil ang sumusunod na figure ay mas sensible na version ng ating PDA:

Sa ngayon, ang ating primary motivation ay hindi para kumita o mag-please sa end users, kundi para sa atin na maging mas productive sa pag-handle ng data at information na makakaharap natin sa ating buhay. Kapag nagsimula ka, ikaw ay

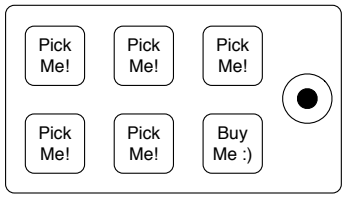

Figure 1.2: Programmers Talking to You

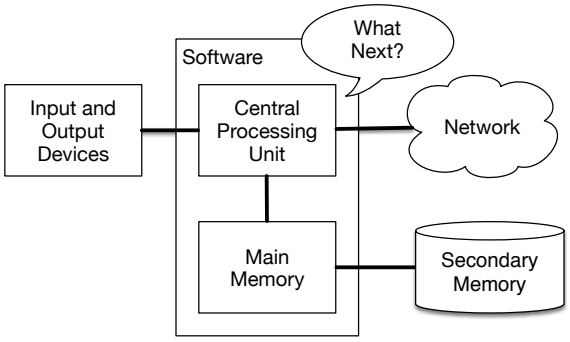

Figure 1.3: Hardware Architecture

pareho ang programmer at ang end user ng iyong programs. Habang nakakakuha ka ng skill bilang programmer at ang programming ay mas naging creative sa iyo, ang iyong mga iniisip ay maaaring mag-turn patungo sa pag-develop ng programs para sa iba.

1.2 Computer hardware architecture

Bago tayo magsimula matuto ng language na sinasabi natin para magbigay ng instructions sa computers para mag-develop ng software, kailangan nating matuto ng kaunti tungkol sa kung paano ginawa ang computers. Kung kukunin mo ang iyong computer o cell phone at titingnan nang malalim sa loob, makikita mo ang sumusunod na parts:

Ang high-level definitions ng mga parts na ito ay ang sumusunod:

- Ang *Central Processing Unit* (o CPU) ay ang parte ng computer na ginawa para maging obsessed sa "ano ang next?" Kung ang iyong computer ay rated sa 3.0 Gigahertz, ibig sabihin ang CPU ay magtatanong ng "What next?" tatlong bilyong beses kada segundo. Kailangan mong matutunan kung paano magsalita nang mabilis para makasabay sa CPU.

- Ang *Main Memory* ay ginagamit para mag-store ng information na kailangan ng CPU nang mabilis. Ang main memory ay halos kasing bilis ng CPU. Pero ang information na naka-store sa main memory ay nawawala kapag ang computer ay naka-off.

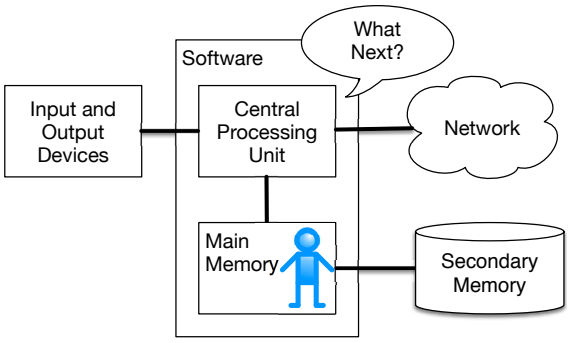

Figure 1.4: Where Are You?

- Ang *Secondary Memory* ay ginagamit din para mag-store ng information, pero mas mabagal ito kaysa sa main memory. Ang advantage ng secondary memory ay maaari itong mag-store ng information kahit walang power ang computer. Mga halimbawa ng secondary memory ay disk drives o flash memory (karaniwang matatagpuan sa USB sticks at portable music players).

- Ang *Input and Output Devices* ay simpleng ating screen, keyboard, mouse, microphone, speaker, touchpad, etc. Sila ang lahat ng paraan kung paano tayo nakikipag-interact sa computer.

- Sa panahon ngayon, karamihan ng computers ay mayroon ding *Network Connection* para mag-retrieve ng information sa network. Maaari nating isipin ang network bilang napakabagal na lugar para mag-store at mag-retrieve ng data na maaaring hindi palaging "up". Kaya sa isang paraan, ang network ay isang mas mabagal at kung minsan unreliable na form ng *Secondary Memory*.

Habang karamihan ng detalye kung paano gumagana ang mga components na ito ay mas mabuting iwanan sa computer builders, nakakatulong na mayroon tayong ilang terminology para makapag-usap tayo tungkol sa mga iba't ibang parts na ito habang sumusulat tayo ng ating programs.

Bilang programmer, ang trabaho mo ay gamitin at i-orchestrate ang bawat isa sa mga resources na ito para solusyonan ang problema na kailangan mong solusyonan at i-analyze ang data na makukuha mo mula sa solusyon. Bilang programmer ay kadalasan mong "kakausapin" ang CPU at sasabihin mo sa kanya kung ano ang gagawin next. Minsan ay sasabihin mo sa CPU na gamitin ang main memory, secondary memory, network, o ang input/output devices.

Kailangan mong maging tao na sumasagot sa tanong ng CPU na "What next?" Pero magiging napaka-uncomfortable na paliitin ka sa 5mm ang taas at ipasok ka sa computer para lang makapag-issue ka ng command na tatlong bilyong beses kada segundo. Kaya sa halip, kailangan mong isulat ang iyong instructions in advance. Tinatawag natin ang mga stored instructions na ito bilang *program* at ang gawain ng pagsusulat ng mga instructions na ito at paggawa ng mga instructions na tama ay *programming*.

1.3 Understanding programming

Sa natitirang bahagi ng librong ito, susubukan naming gawin kang tao na skilled sa art ng programming. Sa huli ay magiging isa kang *programmer* - marahil hindi professional programmer, pero hindi bababa ay magkakaroon ka ng skills para tingnan ang data/information analysis problem at mag-develop ng program para solusyonan ang problema.

Sa isang paraan, kailangan mo ng dalawang skills para maging programmer:

- Una, kailangan mong malaman ang programming language (Python) - kailangan mong malaman ang vocabulary at ang grammar. Kailangan mong makapag-spell ng mga salita sa bagong language na ito nang tama at malaman kung paano gumawa ng well-formed na "sentences" sa bagong language na ito.
- Pangalawa, kailangan mong "mag-kwento". Sa pagsusulat ng kwento, pinagsasama mo ang mga salita at sentences para iparating ang ideya sa reader. Mayroong skill at art sa paggawa ng kwento, at ang skill sa story writing ay napapabuti sa pamamagitan ng pagsusulat at pagkuha ng feedback. Sa programming, ang ating program ay ang "kwento" at ang problema na sinusubukan mong solusyonan ay ang "ideya".

Kapag natutunan mo ang isang programming language tulad ng Python, mas madali mong matututunan ang pangalawang programming language tulad ng JavaScript o C++. Ang bagong programming language ay may napaka-ibang vocabulary at grammar pero ang problem-solving skills ay pareho sa lahat ng programming languages.

Matututunan mo ang "vocabulary" at "sentences" ng Python nang medyo mabilis. Mas matagal bago mo makaya na sumulat ng coherent program para solusyonan ang brand-new problem. Tinuturo namin ang programming katulad ng pagtuturo namin ng writing. Nagsisimula tayo sa pagbasa at pagpapaliwanag ng programs, pagkatapos sumusulat tayo ng simple programs, at pagkatapos sumusulat tayo ng mas kumplikadong programs sa paglipas ng panahon. Sa ilang punto ay "makukuha mo ang iyong muse" at makikita mo ang patterns sa sarili mo at mas natural mong makikita kung paano kumuha ng problema at sumulat ng program na solusyonan ang problemang iyon. At kapag nakarating ka na sa puntong iyon, ang programming ay nagiging napaka-pleasant at creative na proseso.

Nagsisimula tayo sa vocabulary at structure ng Python programs. Maging patient habang ang simple examples ay nagpapaalala sa iyo ng noong nagsimula kang magbasa sa unang pagkakataon.

1.4 Words and sentences

Hindi tulad ng human languages, ang Python vocabulary ay talagang medyo maliit. Tinatawag natin ang "vocabulary" na ito bilang "reserved words" o "keywords". Ang mga salitang ito ay may napaka-espesyal na kahulugan para sa Python. Kapag nakikita ng Python ang mga salitang ito sa isang Python program, mayroon silang

isa at tanging isang kahulugan para sa Python. Mamaya habang sumusulat ka ng programs ay gagawa ka ng sarili mong mga salita na may kahulugan para sa iyo na tinatawag na *variables*. Magkakaroon ka ng malaking kalayaan sa pagpili ng mga pangalan para sa iyong variables, pero hindi mo maaaring gamitin ang alinman sa reserved words ng Python bilang pangalan ng variable.

Kapag nagte-train tayo ng aso, gumagamit tayo ng espesyal na salita tulad ng "sit", "stay", at "fetch". Kapag nakikipag-usap ka sa aso at hindi gumagamit ng alinman sa reserved words, titingin lang sila sa iyo na may quizzical look sa kanilang mukha hanggang sabihin mo ang isang reserved word. Halimbawa, kung sasabihin mo, "I wish more people would walk to improve their overall health", ang marahil na naririnig ng karamihan ng aso ay, "blah blah blah *walk* blah blah blah blah." Iyon ay dahil ang "walk" ay isang reserved word sa dog language. Marami ang maaaring magsabi na ang language sa pagitan ng mga tao at pusa ay walang reserved words[1].

Ang mga reserved words sa language kung saan nakikipag-usap ang mga tao sa Python ay kasama ang sumusunod:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Iyon lang, at hindi tulad ng aso, ang Python ay ganap na trained na. Kapag sinabi mo ang "try", susubukan ng Python tuwing sasabihin mo ito nang walang pagkakamali.

Matututunan natin ang mga reserved words na ito at kung paano sila ginagamit sa tamang panahon, pero sa ngayon ay tututukan natin ang Python equivalent ng "speak" (sa human-to-dog language). Ang maganda sa pagsasabi sa Python na magsalita ay maaari pa nating sabihin sa kanya kung ano ang sasabihin sa pamamagitan ng pagbibigay ng mensahe sa quotes:

```
print('Hello world!')
```

At nakasulat na natin ang ating unang syntactically correct na Python sentence. Ang ating sentence ay nagsisimula sa function na *print* na sinusundan ng string ng text na ating pinili na nakapaloob sa single quotes. Ang mga strings sa print statements ay nakapaloob sa quotes. Ang single quotes at double quotes ay pareho ang ginagawa; karamihan ng tao ay gumagamit ng single quotes maliban sa mga kaso kung saan ang single quote (na apostrophe din) ay lumalabas sa string.

1.5 Conversing with Python

Ngayon na mayroon na tayong salita at simpleng sentence na alam natin sa Python, kailangan nating malaman kung paano magsimula ng conversation sa Python para i-test ang ating bagong language skills.

[1] http://xkcd.com/231/

Bago ka makapag-converse sa Python, kailangan mo munang i-install ang Python software sa iyong computer at matutunan kung paano magsimula ng Python sa iyong computer. Masyadong maraming detalye iyon para sa chapter na ito kaya iminumungkahi ko na konsultahin mo ang www.py4e.com kung saan mayroon akong detailed instructions at screencasts ng pag-setup at pag-start ng Python sa Macintosh at Windows systems. Sa ilang punto, ikaw ay nasa terminal o command window at magta-type ka ng *python* at ang Python interpreter ay magsisimulang mag-execute sa interactive mode at lalabas na medyo ganito:

```
Python 3.11.6 (main, Nov  2 2023, 04:39:43)
   [Clang 14.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Ang >>> prompt ay ang paraan ng Python interpreter na magtanong sa iyo, "Ano ang gusto mong gawin ko next?" Handa na ang Python na makipag-converse sa iyo. Ang kailangan mo lang malaman ay kung paano magsalita ng Python language.

Sabihin natin halimbawa na hindi mo alam kahit ang pinakasimpleng Python language words o sentences. Maaaring gusto mong gamitin ang standard line na ginagamit ng mga astronauts kapag lumapag sila sa malayong planeta at sinusubukan nilang makipag-usap sa mga naninirahan sa planeta:

```
>>> I come in peace, please take me to your leader
File "<stdin>", line 1
    I come in peace take me tou your leader
      ^^^^
SyntaxError: invalid syntax
>>>
```

Hindi masyadong maayos ito. Maliban kung may maisip ka agad, ang mga naninirahan sa planeta ay malamang na tutusukin ka ng kanilang mga sibat, ilalagay ka sa tuhog, iihawin ka sa apoy, at kakainin ka para sa hapunan.

Swerteng dala mo ang kopya ng librong ito sa iyong paglalakbay, at binuksan mo ang pahinang ito at sinubukan mo ulit:

```
>>> print('Hello world!')
Hello world!
```

Mas maayos na ito, kaya sinubukan mong makipag-communicate pa:

```
>>> print('You must be the legendary god that comes from the sky')
You must be the legendary god that comes from the sky
>>> print('We have been waiting for you for a long time')
We have been waiting for you for a long time
>>> print('Our legend says you will be very tasty with mustard')
Our legend says you will be very tasty with mustard
>>> print 'We will have a feast tonight unless you say
```

```
  File "<stdin>", line 1
    print 'We will have a feast tonight unless you say
          ^
SyntaxError: unterminated string literal (detected at line 1)
>>>
```

Ang conversation ay maayos nang sandali at pagkatapos gumawa ka ng pinakamaliit na pagkakamali sa paggamit ng Python language at ibinalik ng Python ang mga sibat.

Sa puntong ito, dapat mo ring mapagtanto na habang ang Python ay napaka complex at powerful at napaka-picky tungkol sa syntax na ginagamit mo para makipag-communicate dito, ang Python ay *hindi* intelligent. Ikaw ay talagang nakikipag-converse lang sa sarili mo, pero gumagamit ng proper syntax.

Sa isang paraan, kapag gumagamit ka ng program na sinulat ng iba, ang conversation ay sa pagitan mo at ng mga programmer na iyon na may Python na nagsisilbing intermediary. Ang Python ay paraan para sa mga creators ng programs na ipahayag kung paano dapat magpatuloy ang conversation. At sa ilang chapters pa lang, ikaw ay magiging isa sa mga programmer na iyon na gumagamit ng Python para makipag-usap sa mga users ng iyong program.

Bago tayo umalis sa ating unang conversation sa Python interpreter, dapat mong malaman ang tamang paraan na sabihin ang "good-bye" kapag nakikipag-interact sa mga naninirahan sa Planet Python:

```
>>> good-bye
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'good' is not defined
>>> if you don't mind, I need to leave
  File "<stdin>", line 1
    if you don't mind, I need to leave
               ^
SyntaxError: unterminated string literal (detected at line 1)
>>> quit()
```

Mapapansin mo na ang error ay iba para sa unang dalawang incorrect attempts. Ang pangalawang error ay iba dahil ang *if* ay isang reserved word at nakita ng Python ang reserved word at naisip na sinusubukan nating sabihin ang isang bagay pero mali ang syntax ng sentence.

Ang tamang paraan na sabihin ang "good-bye" sa Python ay i-enter ang *quit()* sa interactive chevron na >>> prompt. Malamang ay matagal mong hulaan iyon, kaya ang pagkakaroon ng libro na malapit ay malamang na makakatulong.

1.6 Terminology: Interpreter and compiler

Ang Python ay isang *high-level* language na intended na maging relatively straightforward para sa mga tao na basahin at isulat at para sa computers na basahin at

i-process. Ang iba pang high-level languages ay kasama ang Java, C++, PHP, Ruby, Basic, Perl, JavaScript, at marami pa. Ang actual hardware sa loob ng Central Processing Unit (CPU) ay hindi naiintindihan ang alinman sa mga high-level languages na ito.

Ang CPU ay naiintindihan ang language na tinatawag nating *machine language*. Ang machine language ay napaka-simple at totoo lang napaka nakakapagod na isulat dahil ito ay kinakatawan lahat sa zeros at ones:

```
00101000111010010010101000000001111
11100110000011101010010101101101
...
```

Ang machine language ay mukhang medyo simple sa surface, dahil mayroon lang zeros at ones, pero ang syntax nito ay mas complex pa at mas intricate kaysa sa Python. Kaya kakaunti lang ang programmers na sumusulat ng machine language. Sa halip ay gumagawa tayo ng iba't ibang translators para payagan ang mga programmers na sumulat sa high-level languages tulad ng Python o JavaScript at ang mga translators na ito ay nagko-convert ng programs sa machine language para sa actual execution ng CPU.

Dahil ang machine language ay nakatali sa computer hardware, ang machine language ay hindi *portable* sa iba't ibang uri ng hardware. Ang mga programs na sinulat sa high-level languages ay maaaring ilipat sa pagitan ng iba't ibang computers sa pamamagitan ng paggamit ng ibang interpreter sa bagong machine o pag-recompile ng code para gumawa ng machine language version ng program para sa bagong machine.

Ang mga programming language translators na ito ay nahahati sa dalawang general categories: (1) interpreters at (2) compilers.

Ang *interpreter* ay nagbabasa ng source code ng program na sinulat ng programmer, nagpa-parse ng source code, at nag-i-interpret ng instructions on the fly. Ang Python ay isang interpreter at kapag nagpapatakbo tayo ng Python interactively, maaari tayong mag-type ng linya ng Python (isang sentence) at agad na i-process ng Python ito at handa na para sa atin na mag-type ng iba pang linya ng Python.

Ang ilan sa mga linya ng Python ay nagsasabi sa Python na gusto mong tandaan nito ang ilang value para mamaya. Kailangan nating pumili ng pangalan para sa value na iyon para matandaan at maaari nating gamitin ang symbolic name na iyon para i-retrieve ang value mamaya. Ginagamit natin ang term na *variable* para tumukoy sa mga labels na ginagamit natin para tumukoy sa stored data na ito.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

Sa halimbawang ito, hinihiling natin sa Python na tandaan ang value na anim at gamitin ang label na x para ma-retrieve natin ang value mamaya. Sinusuri natin

na talagang naalala ng Python ang value sa pamamagitan ng paggamit ng *print*. Pagkatapos hinihiling natin sa Python na i-retrieve ang x at i-multiply ito sa pito at ilagay ang bagong computed value sa y. Pagkatapos hinihiling natin sa Python na i-print ang value na kasalukuyang nasa y.

Kahit na nagta-type tayo ng mga commands na ito sa Python isa-isa, tinatrato ng Python ang mga ito bilang ordered sequence ng statements na ang mga susunod na statements ay makakakuha ng data na ginawa sa naunang statements. Sumusulat tayo ng ating unang simpleng paragraph na may apat na sentences sa logical at meaningful na order.

Ito ang kalikasan ng *interpreter* na makapagkaroon ng interactive conversation tulad ng ipinakita sa itaas. Ang *compiler* ay kailangan ng buong program sa isang file, at pagkatapos nagpapatakbo ito ng process para i-translate ang high-level source code sa machine language at pagkatapos inilalagay ng compiler ang resulting machine language sa isang file para sa execution mamaya.

Kung mayroon kang Windows system, kadalasan ang mga executable machine language programs na ito ay may suffix na ".exe" o ".dll" na nangangahulugang "executable" at "dynamic link library" ayon sa pagkakabanggit. Sa Linux at Macintosh, walang suffix na natatanging nagmamarka sa file bilang executable.

Kung bubuksan mo ang executable file sa text editor, mukha itong ganap na kakaiba at hindi mabasa:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

Hindi madaling basahin o isulat ang machine language, kaya maganda na mayroon tayong *interpreters* at *compilers* na nagpapahintulot sa atin na sumulat sa high-level languages tulad ng Python o C.

Ngayon sa puntong ito sa ating discussion tungkol sa compilers at interpreters, dapat ay nagtataka ka ng kaunti tungkol sa Python interpreter mismo. Sa anong language ito sinulat? Sinulat ba ito sa compiled language? Kapag nagta-type tayo ng "python", ano talaga ang nangyayari?

Ang Python interpreter ay sinulat sa high-level language na tinatawag na "C". Maaari mong tingnan ang actual source code para sa Python interpreter sa pamamagitan ng pagpunta sa www.python.org at pag-navigate sa kanilang source code. Kaya ang Python ay program mismo at ito ay na-compile sa machine code. Kapag nag-install ka ng Python sa iyong computer (o ang vendor ang nag-install nito), kinopya mo ang machine-code copy ng translated Python program sa iyong system. Sa Windows, ang executable machine code para sa Python mismo ay malamang nasa file na may pangalan tulad ng:

```
C:\Python35\python.exe
```

Iyon ay higit pa sa kailangan mong malaman para maging Python programmer, pero minsan ay nakakatulong na sagutin ang mga maliit na nakakabagot na tanong sa simula pa lang.

1.7 Writing a program

Ang pagta-type ng commands sa Python interpreter ay magandang paraan para mag-experiment sa features ng Python, pero hindi ito inirerekomenda para solusyonan ang mas complex problems.

Kapag gusto nating sumulat ng program, gumagamit tayo ng text editor para isulat ang Python instructions sa isang file, na tinatawag na *script*. Ayon sa convention, ang Python scripts ay may mga pangalan na nagtatapos sa `.py`.

Para i-execute ang script, kailangan mong sabihin sa Python interpreter ang pangalan ng file. Sa command window, magta-type ka ng `python hello.py` tulad ng sumusunod:

```
$ cat hello.py
print('Hello world!')
$ python hello.py
Hello world!
```

Ang "$" ay ang operating system prompt, at ang "cat hello.py" ay nagpapakita sa atin na ang file na "hello.py" ay may one-line Python program para mag-print ng string.

Tinatawag natin ang Python interpreter at sinasabi natin sa kanya na basahin ang source code mula sa file na "hello.py" imbes na mag-prompt sa atin para sa mga linya ng Python code interactively.

Mapapansin mo na hindi kailangan na mayroong *quit()* sa dulo ng Python program sa file. Kapag nagbabasa ang Python ng iyong source code mula sa file, alam nito na huminto kapag naabot na ang dulo ng file.

1.8 What is a program?

Ang definition ng *program* sa pinakabasikong antas ay isang sequence ng Python statements na ginawa para gumawa ng isang bagay. Kahit ang ating simpleng *hello.py* script ay isang program. Ito ay isang one-line program at hindi partikular na useful, pero sa strictest definition, ito ay isang Python program.

Maaaring pinakamadaling maintindihan kung ano ang program sa pamamagitan ng pag-iisip tungkol sa isang problema na maaaring gawin ng program para solusyonan, at pagkatapos tingnan ang isang program na solusyonan ang problemang iyon.

Sabihin natin na gumagawa ka ng Social Computing research sa Facebook posts at interesado ka sa pinaka-frequently used na salita sa isang serye ng posts. Maaari mong i-print ang stream ng Facebook posts at pag-aralan ang text na hinahanap ang pinaka-common na salita, pero magtatagal iyon at napaka-prone sa pagkakamali. Magiging matalino ka kung susulat ka ng Python program para gawin ang task nang mabilis at tumpak para magawa mo ang weekend sa paggawa ng masaya.

Halimbawa, tingnan ang sumusunod na text tungkol sa isang clown at kotse. Tingnan ang text at alamin ang pinaka-common na salita at ilang beses ito nangyari.

```
the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car
```

Pagkatapos isipin na ginagawa mo ang task na ito na tumitingin sa milyun-milyong linya ng text. Totoo lang mas mabilis para sa iyo na matutunan ang Python at sumulat ng Python program para bilangin ang mga salita kaysa sa manually i-scan ang mga salita.

Ang mas magandang balita ay mayroon na akong simpleng program para hanapin ang pinaka-common na salita sa text file. Sinulat ko ito, na-test ko ito, at ngayon ibinibigay ko ito sa iyo para gamitin para makatipid ka ng oras.

```
name = input('Enter file: ')
handle = open(name, 'r')
counts = dict()

for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in list(counts.items()):
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)

# Code: https://www.py4e.com/code3/words.py
```

Hindi mo kailangan na malaman ang Python para gamitin ang program na ito. Kailangan mong makapagdaan sa Chapter 10 ng librong ito para ganap na maintindihan ang awesome Python techniques na ginamit para gawin ang program. Ikaw ang end user, ginagamit mo lang ang program at namamangha sa cleverness nito at kung paano ito nakatipid sa iyo ng napakaraming manual effort. Nagta-type ka lang ng code sa isang file na tinatawag na *words.py* at patakbuhin ito o i-download mo ang source code mula sa http://www.py4e.com/code3/ at patakbuhin ito.

Ito ay magandang halimbawa kung paano ang Python at ang Python language ay nagsisilbing intermediary sa pagitan mo (ang end user) at ako (ang programmer). Ang Python ay paraan para sa atin na magpalitan ng useful instruction sequences (i.e., programs) sa common language na maaaring gamitin ng sinuman na nag-install ng Python sa kanilang computer. Kaya wala sa atin ang nakikipag-usap *sa Python*, sa halip ay nakikipag-communicate tayo sa isa't isa *sa pamamagitan* ng Python.

1.9 The building blocks of programs

Sa susunod na ilang chapters, matututunan natin ang higit pa tungkol sa vocabulary, sentence structure, paragraph structure, at story structure ng Python. Matututunan natin ang powerful capabilities ng Python at kung paano i-compose ang mga capabilities na iyon para gumawa ng useful programs.

Mayroong ilang low-level conceptual patterns na ginagamit natin para gumawa ng programs. Ang mga constructs na ito ay hindi lang para sa Python programs, sila ay parte ng bawat programming language mula sa machine language hanggang sa high-level languages.

input Kumuha ng data mula sa "outside world". Maaari itong pagbasa ng data mula sa isang file, o kahit ilang uri ng sensor tulad ng microphone o GPS. Sa ating initial programs, ang ating input ay manggagaling sa user na nagta-type ng data sa keyboard.

output I-display ang results ng program sa screen o i-store ang mga ito sa isang file o marahil isulat ang mga ito sa device tulad ng speaker para magpatugtog ng musika o magsalita ng text.

sequential execution Gawin ang statements isa-isa sa order na nakita sa script.

conditional execution Suriin ang ilang conditions at pagkatapos i-execute o i-skip ang sequence ng statements.

repeated execution Gawin ang ilang set ng statements nang paulit-ulit, kadalasan may ilang variation.

reuse Sumulat ng set ng instructions minsan at bigyan sila ng pangalan at pagkatapos muling gamitin ang mga instructions na iyon ayon sa pangangailangan sa buong program mo.

Mukhang masyadong simple para maging totoo, at syempre hindi ito kailanman simple. Parang sinasabi na ang paglalakad ay simpleng "paglagay ng isang paa sa harap ng isa pa". Ang "art" ng pagsusulat ng program ay pag-compose at paghabi ng mga basic elements na ito nang maraming beses para gumawa ng isang bagay na useful sa mga users nito.

Ang word counting program sa itaas ay direktang gumagamit ng lahat ng patterns na ito maliban sa isa.

1.10 What could possibly go wrong?

Tulad ng nakita natin sa ating pinakaunang conversations sa Python, kailangan nating makipag-communicate nang napaka-precise kapag sumusulat tayo ng Python code. Ang pinakamaliit na deviation o pagkakamali ay magdudulot sa Python na sumuko sa pagtingin sa iyong program.

Ang mga beginning programmers ay kadalasang iniisip na ang katotohanan na walang lugar ang Python para sa errors ay ebidensya na ang Python ay masama, mapoot, at malupit. Habang ang Python ay mukhang gusto ang lahat ng iba, kilala ng Python sila nang personal at may grudge laban sa kanila. Dahil sa grudge na ito, kinukuha ng Python ang ating perpektong sinulat na programs at tinatanggihan ang mga ito bilang "unfit" para lang pahirapan tayo.

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

>>> primt ('Hello world')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined. Did you mean: 'print'?

>>> I hate you Python!
File "<stdin>", line 1
    I hate you Python!
      ^^^^
SyntaxError: invalid syntax
>>> if you come out of there, I would teach you a lesson
File "<stdin>", line 1
    if you come out of there, I would teach you a lesson
           ^^^^
SyntaxError: invalid syntax
>>>
```

May kaunting makukuha sa pakikipag-argue sa Python. Tool lang ito. Walang emosyon ito at masaya at handa itong maglingkod sa iyo kapag kailangan mo ito. Ang error messages nito ay mukhang harsh, pero mga tawag lang ito ng Python para sa tulong. Tiningnan nito ang iyong na-type, at hindi lang nito naiintindihan ang iyong na-enter.

Ang Python ay mas katulad ng aso, nagmamahal sa iyo nang walang kondisyon, may ilang key words na naiintindihan nito, tumitingin sa iyo na may sweet look sa mukha nito (>>>), at naghihintay na sabihin mo ang isang bagay na naiintindihan nito. Kapag sinabi ng Python ang "SyntaxError: invalid syntax", ito ay nagwawag lang ng buntot at nagsasabi, "Mukhang may sinabi ka pero hindi ko lang naiintindihan ang ibig mong sabihin, pero pakiusap magpatuloy ka sa pakikipagusap sa akin (>>>)."

Habang ang iyong programs ay nagiging mas sophisticated, makakaharap mo ang tatlong general types ng errors:

Syntax errors Ito ang unang errors na gagawin mo at pinakamadaling ayusin. Ang syntax error ay nangangahulugan na nilabag mo ang "grammar" rules ng Python. Ginagawa ng Python ang makakaya nito para ituro ang linya at character kung saan napansin nito na nalito ito. Ang nakakalito na parte ng syntax errors ay kung minsan ang pagkakamali na kailangang ayusin ay talagang mas maaga sa program kaysa sa kung saan *napansin* ng Python na nalito ito. Kaya ang linya at character na itinuturo ng Python sa syntax error ay maaaring starting point lang para sa iyong investigation.

Logic errors Ang logic error ay kapag ang iyong program ay may magandang syntax pero may pagkakamali sa order ng statements o marahil pagkakamali sa kung paano nauugnay ang statements sa isa't isa. Magandang halimbawa

ng logic error ay maaaring, “uminom mula sa water bottle mo, ilagay ito sa backpack mo, maglakad patungo sa library, at pagkatapos ibalik ang takip sa bottle.”

Semantic errors Ang semantic error ay kapag ang iyong description ng mga hakbang na gagawin ay syntactically perfect at nasa tamang order, pero may simpleng pagkakamali sa program. Ang program ay perpektong tama pero hindi ito ginagawa ang gusto mong *intended* na gawin nito. Simpleng halimbawa ay kung nagbibigay ka ng direksyon sa isang tao patungo sa isang restaurant at sinabi mo, “. . . kapag naabot mo ang intersection na may gas station, lumiko sa kaliwa at pumunta ng isang milya at ang restaurant ay pulang building sa iyong kaliwa.” Ang kaibigan mo ay napaka-late at tumawag sa iyo para sabihin na nasa bukid sila at naglalakad sa likod ng kamalig, walang sign ng restaurant. Pagkatapos sinabi mo “lumiko ka ba sa kaliwa o kanan sa gas station?” at sinabi nila, “Sinunod ko nang perpekto ang direksyon mo, nakasulat ko sila, sinasabi nito na lumiko sa kaliwa at pumunta ng isang milya sa gas station.” Pagkatapos sinabi mo, “Pasensya na, dahil habang ang instructions ko ay syntactically correct, nakakalungkot na naglalaman sila ng maliit pero hindi natuklasang semantic error.”.

Muli sa lahat ng tatlong uri ng errors, ang Python ay sinusubukan lang ang makakaya nito para gawin nang eksakto ang hiniling mo.

1.11 Debugging

Kapag nag-spit out ang Python ng error o kahit kapag binigyan ka nito ng result na iba sa gusto mo, pagkatapos ay magsisimula ang paghahanap ng sanhi ng error. Ang Debugging ay ang proseso ng paghahanap ng sanhi ng error sa iyong code. Kapag nagde-debug ka ng program, at lalo na kung nagtatrabaho ka sa hard bug, may apat na bagay na subukan:

reading Suriin ang iyong code, basahin ito sa sarili mo, at suriin na sinasabi nito ang gusto mong sabihin.

running Mag-experiment sa pamamagitan ng paggawa ng changes at pagpapatakbo ng iba’t ibang versions. Kadalasan kung i-display mo ang tamang bagay sa tamang lugar sa program, ang problema ay nagiging obvious, pero minsan kailangan mong gumugol ng ilang oras para gumawa ng scaffolding.

ruminating Maglaan ng oras para mag-isip! Anong uri ng error ito: syntax, runtime, semantic? Anong information ang makukuha mo mula sa error messages, o mula sa output ng program? Anong uri ng error ang maaaring maging sanhi ng problemang nakikita mo? Ano ang huling binago mo, bago lumabas ang problema?

retreating Sa ilang punto, ang pinakamabuting gawin ay umatras, i-undo ang mga kamakailang changes, hanggang makabalik ka sa program na gumagana at naiintindihan mo. Pagkatapos maaari mong simulan ang muling pagbuo.

Ang mga beginning programmers ay minsan natatrap sa isa sa mga activities na ito at nakakalimutan ang iba. Ang paghahanap ng hard bug ay nangangailangan ng reading, running, ruminating, at minsan retreating. Kung natatrap ka sa isa

sa mga activities na ito, subukan ang iba. Bawat activity ay may sariling failure mode.

Halimbawa, ang pagbabasa ng iyong code ay maaaring makatulong kung ang problema ay typographical error, pero hindi kung ang problema ay conceptual misunderstanding. Kung hindi mo naiintindihan ang ginagawa ng iyong program, maaari mong basahin ito ng 100 beses at hindi makita ang error, dahil ang error ay nasa ulo mo.

Ang pagpapatakbo ng experiments ay maaaring makatulong, lalo na kung nagpapatakbo ka ng maliit, simpleng tests. Pero kung nagpapatakbo ka ng experiments nang hindi nag-iisip o nagbabasa ng iyong code, maaari kang mahulog sa pattern na tinatawag kong "random walk programming", na siyang proseso ng paggawa ng random changes hanggang gawin ng program ang tamang bagay. Hindi na kailangang sabihin, ang random walk programming ay maaaring tumagal ng mahabang panahon.

Kailangan mong maglaan ng oras para mag-isip. Ang Debugging ay parang experimental science. Dapat mayroon kang hindi bababa sa isang hypothesis tungkol sa kung ano ang problema. Kung mayroong dalawa o higit pang posibilidad, subukan mong mag-isip ng test na magtatanggal sa isa sa kanila.

Ang pagkuha ng break ay nakakatulong sa pag-iisip. Ganun din ang pakikipagusap. Kung ipapaliwanag mo ang problema sa iba (o kahit sa sarili mo), minsan ay makikita mo ang sagot bago mo matapos ang pagtatanong.

Pero kahit ang pinakamahusay na debugging techniques ay mabibigo kung mayroong napakaraming errors, o kung ang code na sinusubukan mong ayusin ay masyadong malaki at kumplikado. Minsan ang pinakamabuting option ay umatras, gawing simple ang program hanggang makarating ka sa isang bagay na gumagana at naiintindihan mo.

Ang mga beginning programmers ay kadalasang ayaw umatras dahil hindi nila kayang tanggalin ang isang linya ng code (kahit mali ito). Kung makakaramdam ka ng mas mabuti, kopyahin ang iyong program sa ibang file bago mo simulan ang pagtatanggal. Pagkatapos maaari mong i-paste ang mga piraso pabalik nang paunti-unti sa bawat pagkakataon.

1.12 The learning journey

Habang nagpapatuloy ka sa natitirang bahagi ng libro, huwag matakot kung ang mga concepts ay hindi mukhang magkasya nang maayos sa unang pagkakataon. Noong natututo kang magsalita, hindi ito problema sa unang ilang taon mo na gumagawa ka lang ng cute na gurgling noises. At OK lang kung tumagal ng anim na buwan para sa iyo na lumipat mula sa simpleng vocabulary patungo sa simpleng sentences at tumagal ng 5-6 taon pa para lumipat mula sa sentences patungo sa paragraphs, at ilang taon pa para makaya mong sumulat ng interesting na kumpletong short story sa sarili mo.

Gusto naming matutunan mo ang Python nang mas mabilis, kaya tinuturo namin ito lahat sa parehong panahon sa susunod na ilang chapters. Pero parang pag-aaral ng bagong language na nangangailangan ng oras para ma-absorb at maintindihan bago ito pakiramdam natural. Nagdudulot ito ng ilang kalituhan habang binibisita

at muling binibisita namin ang mga topics para subukan na makita mo ang big picture habang tinutukoy namin ang maliliit na fragments na bumubuo sa big picture na iyon. Habang ang libro ay sinulat linearly, at kung kumukuha ka ng course ay magpapatuloy ito sa linear fashion, huwag mag-atubiling maging napaka-nonlinear sa kung paano mo nilalapitan ang material. Tumingin sa harap at likod at magbasa nang may light touch. Sa pamamagitan ng pag-skim ng mas advanced na material nang hindi ganap na naiintindihan ang mga detalye, makakakuha ka ng mas mahusay na pag-unawa sa "bakit?" ng programming. Sa pamamagitan ng pagre-review ng naunang material at kahit paggawa ulit ng naunang exercises, mapapagtanto mo na talagang marami kang natutunang material kahit na ang material na kasalukuyang tinitingnan mo ay mukhang medyo hindi maunawaan.

Kadalasan kapag natututo ka ng iyong unang programming language, mayroong ilang magagandang "Ah Hah!" moments kung saan maaari kang tumingin mula sa pagpukpok sa ilang bato gamit ang martilyo at pait at umalis at makita na talagang gumagawa ka ng magandang eskultura.

Kung ang isang bagay ay mukhang partikular na mahirap, kadalasan walang halaga sa pagpupuyat buong gabi at pagtingin dito. Magpahinga, matulog, kumain ng meryenda, ipaliwanag ang problema mo sa isang tao (o marahil sa iyong aso), at pagkatapos bumalik dito na may fresh eyes. Sinisiguro ko sa iyo na kapag natutunan mo ang programming concepts sa libro ay titingnan mo pabalik at makikita mo na talagang madali at elegant ang lahat at kinailangan mo lang ng kaunting oras para ma-absorb ito.

1.13 Glossary

bug Error sa isang program.

central processing unit Ang puso ng anumang computer. Ito ang nagpapatakbo ng software na sinulat natin; tinatawag din na "CPU" o "the processor".

compile I-translate ang program na sinulat sa high-level language sa low-level language nang sabay-sabay, bilang paghahanda para sa execution mamaya.

high-level language Programming language tulad ng Python na idinisenyo para madaling basahin at isulat ng mga tao.

interactive mode Paraan ng paggamit ng Python interpreter sa pamamagitan ng pagta-type ng commands at expressions sa prompt.

interpret I-execute ang program sa high-level language sa pamamagitan ng pag-translate nito isa isang linya sa bawat pagkakataon.

low-level language Programming language na idinisenyo para madaling i-execute ng computer; tinatawag din na "machine code" o "assembly language".

machine code Ang pinakamababang-level na language para sa software, na siyang language na direktang na-e-execute ng central processing unit (CPU).

main memory Nag-i-store ng programs at data. Ang main memory ay nawawala ang information kapag ang power ay naka-off.

parse Suriin ang program at i-analyze ang syntactic structure.

portability Property ng program na maaaring tumakbo sa higit sa isang uri ng computer.

print function Instruction na nagdudulot sa Python interpreter na mag-display ng value sa screen.

problem solving Ang proseso ng pagbuo ng problema, paghahanap ng solusyon, at pagpapahayag ng solusyon.

program Set ng instructions na tumutukoy sa computation.

prompt Kapag ang program ay nagdi-display ng mensahe at nagpa-pause para sa user na mag-type ng ilang input sa program.

secondary memory Nag-i-store ng programs at data at nagpapanatili ng information kahit ang power ay naka-off. Sa pangkalahatan ay mas mabagal kaysa sa main memory. Mga halimbawa ng secondary memory ay kasama ang disk drives at flash memory sa USB sticks.

semantics Ang kahulugan ng isang program.

semantic error Error sa program na nagdudulot dito na gumawa ng iba sa gusto ng programmer.

source code Program sa high-level language.

1.14 Exercises

Exercise 1: Ano ang function ng secondary memory sa computer?

a) I-execute ang lahat ng computation at logic ng program
b) I-retrieve ang web pages sa Internet
c) Mag-store ng information para sa long term, kahit lampas sa power cycle
d) Kumuha ng input mula sa user

Exercise 2: Ano ang program?

Exercise 3: Ano ang pagkakaiba sa pagitan ng compiler at interpreter?

Exercise 4: Alin sa sumusunod ang naglalaman ng “machine code”?

a) Ang Python interpreter
b) Ang keyboard
c) Python source file
d) Word processing document

Exercise 5: Ano ang mali sa sumusunod na code:

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

Exercise 6: Saan sa computer naka-store ang variable tulad ng “x” pagkatapos matapos ang sumusunod na Python line?

```
x = 123
```

a) Central processing unit
b) Main Memory
c) Secondary Memory
d) Input Devices
e) Output Devices

Exercise 7: Ano ang i-print ng sumusunod na program:

```
x = 43
x = x - 1
print(x)
```

a) 43
b) 42
c) x - 1
d) Error dahil ang x = x - 1 ay hindi posible mathematically

Exercise 8: Ipaliwanag ang bawat isa sa sumusunod gamit ang halimbawa ng human capability: (1) Central processing unit, (2) Main Memory, (3) Secondary Memory, (4) Input Device, at (5) Output Device. Halimbawa, “Ano ang human equivalent ng Central Processing Unit”?

Chapter 2

Variables, expressions, at statements

2.1 Values and types

Ang *value* ay isa sa mga basic na bagay na ginagampanan ng program, tulad ng letra o numero. Ang mga values na nakita natin hanggang ngayon ay 1, 2, at "Hello, World!"

Ang mga values na ito ay kabilang sa iba't ibang *types*: ang 2 ay integer, at ang "Hello, World!" ay *string*, kaya tinatawag na ganito dahil naglalaman ito ng "string" ng mga letra. Ikaw (at ang interpreter) ay makakakilala ng strings dahil nakapaloob sila sa quotation marks.

Ang `print` statement ay gumagana din para sa integers. Ginagamit natin ang `python` command para simulan ang interpreter.

```
python
>>> print(4)
4
```

Kung hindi ka sigurado kung anong type ang value, maaaring sabihin sa iyo ng interpreter.

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
```

Hindi nakakagulat, ang strings ay kabilang sa type na `str` at ang integers ay kabilang sa type na `int`. Mas hindi halata, ang mga numero na may decimal point ay kabilang sa type na tinatawag na `float`, dahil ang mga numerong ito ay kinakatawan sa format na tinatawag na *floating point*.

```
>>> type(3.2)
<class 'float'>
```

Paano naman ang mga values tulad ng “17” at “3.2”? Mukha silang mga numero, pero nakapaloob sila sa quotation marks tulad ng strings.

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

Mga strings sila.

Kapag nagta-type ka ng malaking integer, maaari kang matukso na gumamit ng commas sa pagitan ng mga grupo ng tatlong digits, tulad ng 1,000,000. Hindi ito legal integer sa Python, pero legal ito:

```
>>> print(1,000,000)
1 0 0
```

Well, hindi iyon ang inaasahan natin! Binibigyang-kahulugan ng Python ang 1,000,000 bilang comma-separated sequence ng integers, na ini-print nito na may spaces sa pagitan.

Ito ang unang halimbawa na nakita natin ng semantic error: ang code ay tumatakbo nang hindi gumagawa ng error message, pero hindi ito gumagawa ng “tamang” bagay.

2.2 Variables

Isa sa pinakamakapangyarihang features ng programming language ay ang kakayahang manipulahin ang *variables*. Ang variable ay pangalan na tumutukoy sa isang value.

Ang *assignment statement* ay gumagawa ng bagong variables at binibigyan sila ng values:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
```

Ang halimbawang ito ay gumagawa ng tatlong assignments. Ang una ay nag-a-assign ng string sa isang bagong variable na may pangalang `message`; ang pangalawa ay nag-a-assign ng integer na 17 sa `n`; ang pangatlo ay nag-a-assign ng (approximate) value ng π sa `pi`.

Para i-display ang value ng variable, maaari mong gamitin ang print statement:

```
>>> print(n)
17
>>> print(pi)
3.141592653589793
```

Ang type ng variable ay ang type ng value na tinutukoy nito.

```
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

2.3 Variable names and keywords

Ang mga programmers ay karaniwang pumipili ng mga pangalan para sa kanilang variables na makabuluhan at nagdo-document kung para saan ginagamit ang variable.

Ang mga variable names ay maaaring arbitrary na mahaba. Maaari silang maglalaman ng parehong letra at numero, pero hindi sila maaaring magsimula sa numero. Legal na gumamit ng uppercase letters, pero magandang ideya na magsimula ang variable names sa lowercase letter (makikita mo kung bakit mamaya).

Ang underscore character (_) ay maaaring lumabas sa pangalan. Kadalasan itong ginagamit sa mga pangalan na may maraming salita, tulad ng `my_name` o `airspeed_of_unladen_swallow`. Ang mga variable names ay maaaring magsimula sa underscore character, pero karaniwang iniiwasan natin ito maliban kung sumusulat tayo ng library code para sa iba na gamitin.

Kung bibigyan mo ang variable ng illegal na pangalan, makakakuha ka ng syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Ang `76trombones` ay illegal dahil nagsisimula ito sa numero. Ang `more@` ay illegal dahil naglalaman ito ng illegal character, @. Pero ano ang mali sa `class`?

Lumalabas na ang `class` ay isa sa mga *keywords* ng Python. Ang interpreter ay gumagamit ng keywords para makilala ang structure ng program, at hindi sila maaaring gamitin bilang variable names.

Ang Python ay nagre-reserve ng 35 keywords:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Maaaring gusto mong panatilihin ang list na ito na malapit. Kung nagre-reklamo ang interpreter tungkol sa isa sa iyong variable names at hindi mo alam kung bakit, tingnan kung nasa list na ito.

2.4 Statements

Ang *statement* ay unit ng code na maaaring i-execute ng Python interpreter. Nakita na natin ang dalawang uri ng statements: ang print bilang expression statement at assignment.

Kapag nagta-type ka ng statement sa interactive mode, i-e-execute ito ng interpreter at i-di-display ang result, kung mayroon.

Ang script ay karaniwang naglalaman ng sequence ng statements. Kung mayroong higit sa isang statement, ang results ay lumalabas isa-isa habang na-e-execute ang statements.

Halimbawa, ang script

```
print(1)
x = 2
print(x)
```

ay gumagawa ng output

```
1
2
```

Ang assignment statement ay hindi gumagawa ng output.

2.5 Operators and operands

Ang *Operators* ay espesyal na symbols na kumakatawan sa computations tulad ng addition at multiplication. Ang mga values na ina-applyan ng operator ay tinatawag na *operands*.

Ang mga operators na `+`, `-`, `*`, `/`, at `**` ay gumagawa ng addition, subtraction, multiplication, division, at exponentiation, tulad sa sumusunod na halimbawa:

```
20+32
hour-1
hour*60+minute
minute/60
5**2
(5+9)*(15-7)
```

May pagbabago sa division operator sa pagitan ng Python 2 at Python 3. Sa Python 3, ang result ng division na ito ay floating point result:

```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

Ang division operator sa Python 2 ay magdi-divide ng dalawang integers at i-truncate ang result sa integer:

```
>>> minute = 59
>>> minute/60
0
```

Para makuha ang parehong sagot sa Python 3 gumamit ng floored (`//` integer) division.

```
>>> minute = 59
>>> minute//60
0
```

Sa Python 3 ang integer division ay gumagana nang mas katulad ng inaasahan mo kung nag-enter ka ng expression sa calculator.

2.6 Expressions

Ang *expression* ay kombinasyon ng values, variables, at operators. Ang value na mag-isa ay itinuturing na expression, at ganun din ang variable, kaya ang sumusunod ay lahat legal expressions (assuming na ang variable na `x` ay na-assign na ng value):

```
17
x
x + 17
```

Kung magta-type ka ng expression sa interactive mode, i-*evaluate* ito ng interpreter at i-di-display ang result:

```
>>> 1 + 1
2
```

Pero sa script, ang expression na mag-isa ay walang ginagawa! Ito ay karaniwang pinagmumulan ng kalituhan para sa mga beginners.

Exercise 1: I-type ang sumusunod na statements sa Python interpreter para makita kung ano ang ginagawa nila:

```
5
x = 5
x + 1
```

2.7 Order of operations

Kapag higit sa isang operator ang lumalabas sa expression, ang order ng evaluation ay depende sa *rules of precedence*. Para sa mathematical operators, ang Python ay sumusunod sa mathematical convention. Ang acronym na *PEMDAS* ay kapaki-pakinabang na paraan para matandaan ang mga rules:

- Ang *P*arentheses ay may pinakamataas na precedence at maaaring gamitin para pilitin ang expression na i-evaluate sa order na gusto mo. Dahil ang expressions sa parentheses ay na-e-evaluate muna, ang `2 * (3-1)` ay 4, at ang `(1+1)**(5-2)` ay 8. Maaari mo ring gamitin ang parentheses para gawing mas madaling basahin ang expression, tulad sa `(minute * 100) / 60`, kahit hindi nito binabago ang result.
- Ang *E*xponentiation ay may susunod na pinakamataas na precedence, kaya ang `2**1+1` ay 3, hindi 4, at ang `3*1**3` ay 3, hindi 27.
- Ang *M*ultiplication at *D*ivision ay may parehong precedence, na mas mataas kaysa sa *A*ddition at *S*ubtraction, na may parehong precedence din. Kaya ang `2*3-1` ay 5, hindi 4, at ang `6+4/2` ay 8, hindi 5.
- Ang mga operators na may parehong precedence ay na-e-evaluate mula kaliwa hanggang kanan. Kaya ang expression na `5-3-1` ay 1, hindi 3, dahil ang `5-3` ay nangyayari muna at pagkatapos ang `1` ay ibabawas mula sa 2.

Kapag may duda, palaging maglagay ng parentheses sa iyong expressions para masiguro na ang computations ay ginagawa sa order na gusto mo.

2.8 Modulus operator

Ang *modulus operator* ay gumagana sa integers at nagbibigay ng remainder kapag ang unang operand ay hinati sa pangalawa. Sa Python, ang modulus operator ay percent sign (`%`). Ang syntax ay pareho sa iba pang operators:

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

Kaya ang 7 na hinati sa 3 ay 2 na may natitirang 1.

Ang modulus operator ay naging kapaki-pakinabang. Halimbawa, maaari mong suriin kung ang isang numero ay divisible sa isa pa: kung ang `x % y` ay zero, pagkatapos ang `x` ay divisible sa `y`.

Maaari mo ring kunin ang right-most digit o digits mula sa numero. Halimbawa, ang `x % 10` ay nagbibigay ng right-most digit ng `x` (sa base 10). Katulad nito, ang `x % 100` ay nagbibigay ng huling dalawang digits.

2.9 String operations

Ang `+` operator ay gumagana sa strings, pero hindi ito addition sa mathematical sense. Sa halip ay gumagawa ito ng *concatenation*, na nangangahulugang pag-uugnay ng strings sa pamamagitan ng pag-link sa kanila mula dulo hanggang dulo. Halimbawa:

```
>>> first = 10
>>> second = 15
>>> print(first+second)
25
>>> first = '100'
>>> second = '150'
>>> print(first + second)
100150
```

Ang `*` operator ay gumagana din sa strings sa pamamagitan ng pag-multiply ng content ng string sa integer. Halimbawa:

```
>>> first = 'Test '
>>> second = 3
>>> print(first * second)
Test Test Test
```

2.10 Asking the user for input

Minsan gusto nating kunin ang value para sa variable mula sa user sa pamamagitan ng kanilang keyboard. Ang Python ay nagbibigay ng built-in function na tinatawag na `input` na kumukuha ng input mula sa keyboard[1]. Kapag tinawag ang function na ito, ang program ay humihinto at naghihintay na mag-type ang user ng isang bagay. Kapag pinindot ng user ang `Return` o `Enter`, ang program ay nagpapatuloy at ang `input` ay nagre-return ng na-type ng user bilang string.

```
>>> inp = input()
Some silly stuff
>>> print(inp)
Some silly stuff
```

[1] Sa Python 2, ang function na ito ay may pangalang `raw_input`.

Bago kumuha ng input mula sa user, magandang ideya na mag-print ng prompt na nagsasabi sa user kung ano ang i-input. Maaari mong ipasa ang string sa `input` para i-display sa user bago mag-pause para sa input:

```
>>> name = input('What is your name?\n')
What is your name?
Chuck
>>> print(name)
Chuck
```

Ang sequence na `\n` sa dulo ng prompt ay kumakatawan sa *newline*, na espesyal na character na nagdudulot ng line break. Iyon ang dahilan kung bakit ang input ng user ay lumalabas sa ibaba ng prompt.

Kung inaasahan mong mag-type ang user ng integer, maaari mong subukan na i-convert ang return value sa `int` gamit ang `int()` function:

```
>>> prompt = 'What...is the airspeed velocity of an unladen swallow?\n'
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
17
>>> int(speed)
17
>>> int(speed) + 5
22
```

Pero kung ang user ay nag-type ng iba sa string ng digits, makakakuha ka ng error:

```
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with
base 10: 'What do you mean, an African or a European swallow?'
```

Makikita natin kung paano haharapin ang ganitong uri ng error mamaya.

2.11 Comments

Habang ang mga programs ay nagiging mas malaki at mas kumplikado, nagiging mas mahirap silang basahin. Ang formal languages ay dense, at kadalasang mahirap tingnan ang isang piraso ng code at alamin kung ano ang ginagawa nito, o bakit.

Para sa dahilang ito, magandang ideya na magdagdag ng notes sa iyong programs para ipaliwanag sa natural language kung ano ang ginagawa ng program. Ang mga notes na ito ay tinatawag na *comments*, at sa Python nagsisimula sila sa `#` symbol:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

Sa kasong ito, ang comment ay lumalabas sa sarili nitong linya. Maaari mo ring ilagay ang comments sa dulo ng linya:

```
percentage = (minute * 100) / 60      # percentage of an hour
```

Lahat mula sa # hanggang sa dulo ng linya ay hindi pinapansin; walang epekto ito sa program.

Ang comments ay pinaka-kapaki-pakinabang kapag nagdo-document sila ng hindi halatang features ng code. Makatwiran na i-assume na ang reader ay makakapag-figure out kung *ano* ang ginagawa ng code; mas kapaki-pakinabang na ipaliwanag ang *bakit*.

Ang comment na ito ay redundant sa code at walang silbi:

```
v = 5      # assign 5 to v
```

Ang comment na ito ay naglalaman ng kapaki-pakinabang na information na wala sa code:

```
v = 5      # velocity in meters/second.
```

Ang magagandang variable names ay maaaring bawasan ang pangangailangan para sa comments, pero ang mahahabang pangalan ay maaaring gawing mahirap basahin ang complex expressions, kaya may trade-off.

2.12 Choosing mnemonic variable names

Hangga't sinusunod mo ang simpleng rules ng variable naming, at iniiwasan ang reserved words, marami kang pagpipilian kapag pinangalanan mo ang iyong variables. Sa simula, ang pagpipiliang ito ay maaaring nakakalito kapag nagbabasa ka ng program at kapag sumusulat ka ng sarili mong programs. Halimbawa, ang sumusunod na tatlong programs ay magkapareho sa kung ano ang nagagawa nila, pero napaka iba kapag binabasa mo sila at sinusubukan mong maintindihan.

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

```
x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

Ang Python interpreter ay nakikita ang lahat ng tatlong programs na ito bilang *eksaktong pareho* pero ang mga tao ay nakikita at naiintindihan ang mga programs na ito nang medyo iba. Ang mga tao ay pinakamabilis na maiintindihan ang *intent* ng pangalawang program dahil ang programmer ay pumili ng variable names na sumasalamin sa kanilang intent tungkol sa kung ano ang data na i-store sa bawat variable.

Tinatawag natin ang mga matalinong napiling variable names na "mnemonic variable names". Ang salitang *mnemonic*[2] ay nangangahulugang "memory aid". Pumipili tayo ng mnemonic variable names para matulungan tayong matandaan kung bakit natin ginawa ang variable sa simula pa lang.

Habang ang lahat ng ito ay mukhang maganda, at napakagandang ideya na gumamit ng mnemonic variable names, ang mnemonic variable names ay maaaring makagambala sa kakayahan ng beginning programmer na i-parse at maintindihan ang code. Ito ay dahil ang mga beginning programmers ay hindi pa na-memorize ang reserved words (may 35 lang sa kanila) at minsan ang mga variables na may mga pangalan na masyadong descriptive ay nagsisimulang magmukhang parte ng language at hindi lang well-chosen variable names.

Tingnan mo ang sumusunod na Python sample code na naglo-loop sa ilang data. Tatalakayin natin ang loops sa lalong madaling panahon, pero sa ngayon subukan lang na alamin kung ano ang ibig sabihin nito:

```
for word in words:
    print(word)
```

Ano ang nangyayari dito? Alin sa mga tokens (for, word, in, etc.) ang reserved words at alin ang variable names lang? Naiintindihan ba ng Python sa fundamental level ang konsepto ng words? Ang mga beginning programmers ay may problema sa paghihiwalay kung aling parts ng code ang *dapat* pareho sa halimbawang ito at aling parts ng code ang simpleng choices lang ng programmer.

Ang sumusunod na code ay katumbas ng code sa itaas:

```
for slice in pizza:
    print(slice)
```

Mas madali para sa beginning programmer na tingnan ang code na ito at malaman kung aling parts ang reserved words na tinukoy ng Python at aling parts ang simpleng variable names lang na pinili ng programmer. Medyo malinaw na walang fundamental understanding ang Python tungkol sa pizza at slices at sa katotohanan na ang pizza ay binubuo ng set ng isa o higit pang slices.

[2] Tingnan ang https://en.wikipedia.org/wiki/Mnemonic para sa extended description ng salitang "mnemonic".

Pero kung ang program natin ay talagang tungkol sa pagbasa ng data at paghahanap ng words sa data, ang `pizza` at `slice` ay napaka-un-mnemonic variable names. Ang pagpili sa kanila bilang variable names ay nakakagambala sa kahulugan ng program.

Pagkatapos ng medyo maikling panahon, malalaman mo ang pinakakaraniwang reserved words at magsisimula kang makita ang reserved words na tumatalon sa iyo:

Ang mga parts ng code na tinukoy ng Python (`for`, `in`, `print`, at `:`) ay naka-bold at ang programmer-chosen variables (`word` at `words`) ay hindi naka-bold. Maraming text editors ang aware sa Python syntax at magko-color ng reserved words nang iba para bigyan ka ng clues para panatilihing hiwalay ang iyong variables at reserved words. Pagkatapos ng ilang panahon magsisimula kang magbasa ng Python at mabilis na matukoy kung ano ang variable at kung ano ang reserved word.

2.13 Debugging

Sa puntong ito, ang syntax error na malamang mong gawin ay illegal variable name, tulad ng `class` at `yield`, na keywords, o `odd~job` at `US$`, na naglalaman ng illegal characters.

Kung maglalagay ka ng space sa variable name, iniisip ng Python na dalawang operands ito na walang operator:

```
>>> bad name = 5
SyntaxError: invalid syntax
```

Para sa syntax errors, ang error messages ay hindi masyadong nakakatulong. Ang pinakakaraniwang messages ay `SyntaxError: invalid syntax` na hindi masyadong informative.

Ang runtime error na malamang mong gawin ay "use before def;" iyon ay, sinusubukan mong gamitin ang variable bago mo i-assign ang value. Maaari itong mangyari kung mali ang spelling mo ng variable name:

```
>>> principal = 327.68
>>> interest = principle * rate
NameError: name 'principle' is not defined
```

Ang mga variable names ay case sensitive, kaya ang `LaTeX` ay hindi pareho sa `latex`.

Sa puntong ito, ang pinakamalamang na sanhi ng semantic error ay ang order ng operations. Halimbawa, para i-evaluate ang $1/2\pi$, maaari kang matukso na sumulat

```
>>> 1.0 / 2.0 * pi
```

Pero ang division ay nangyayari muna, kaya makakakuha ka ng $\pi/2$, na hindi pareho! Walang paraan para sa Python na malaman kung ano ang ibig mong sabihin, kaya sa kasong ito hindi ka makakakuha ng error message; mali lang ang sagot na makukuha mo.

2.14 Glossary

assignment Statement na nag-a-assign ng value sa variable.
concatenate Pag-uugnay ng dalawang operands mula dulo hanggang dulo.
comment Information sa program na para sa iba pang programmers (o sinumang nagbabasa ng source code) at walang epekto sa execution ng program.
evaluate Pagpapasimple ng expression sa pamamagitan ng paggawa ng operations sa order para magbigay ng isang value.
expression Kombinasyon ng variables, operators, at values na kumakatawan sa isang result value.
floating point Type na kumakatawan sa mga numero na may fractional parts.
integer Type na kumakatawan sa whole numbers.
keyword Reserved word na ginagamit ng compiler para i-parse ang program; hindi mo maaaring gamitin ang keywords tulad ng `if`, `def`, at `while` bilang variable names.
mnemonic Memory aid. Kadalasang binibigyan natin ang variables ng mnemonic names para matulungan tayo na matandaan kung ano ang naka-store sa variable.
modulus operator Operator, na tinutukoy ng percent sign (`%`), na gumagana sa integers at nagbibigay ng remainder kapag ang isang numero ay hinati sa isa pa.
operand Isa sa mga values na ginagampanan ng operator.
operator Espesyal na symbol na kumakatawan sa simpleng computation tulad ng addition, multiplication, o string concatenation.
rules of precedence Set ng rules na namamahala sa order kung saan ang expressions na may maraming operators at operands ay na-e-evaluate.
statement Section ng code na kumakatawan sa command o action. Hanggang ngayon, ang statements na nakita natin ay assignments at print expression statement.
string Type na kumakatawan sa sequences ng characters.
type Kategorya ng values. Ang mga types na nakita natin hanggang ngayon ay integers (type `int`), floating-point numbers (type `float`), at strings (type `str`).
value Isa sa mga basic units ng data, tulad ng numero o string, na ginagampanan ng program.
variable Pangalan na tumutukoy sa value.

2.15 Exercises

Exercise 2: Sumulat ng program na gumagamit ng `input` para mag-prompt sa user para sa kanilang pangalan at pagkatapos ay batiin sila.

```
Enter your name: Chuck
Hello Chuck
```

Exercise 3: Sumulat ng program para mag-prompt sa user para sa hours at rate per hour para i-compute ang gross pay.

```
Enter Hours: 35
```

```
Enter Rate: 2.75
Pay: 96.25
```

Hindi natin iintindihin ang pagtiyak na ang pay natin ay may eksaktong dalawang digits pagkatapos ng decimal place sa ngayon. Kung gusto mo, maaari mong subukan ang built-in Python `round` function para maayos na i-round ang resulting pay sa dalawang decimal places.

Exercise 4: I-assume na i-e-execute natin ang sumusunod na assignment statements:

```
width = 17
height = 12.0
```

Para sa bawat isa sa sumusunod na expressions, isulat ang value ng expression at ang type (ng value ng expression).

1. `width//2`
2. `width/2.0`
3. `height/3`
4. `1 + 2 * 5`

Gamitin ang Python interpreter para suriin ang iyong mga sagot.

Exercise 5: Sumulat ng program na nagpo-prompt sa user para sa Celsius temperature, i-convert ang temperature sa Fahrenheit, at i-print ang converted temperature.

Chapter 3

Conditional execution

3.1 Boolean expressions

Ang *boolean expression* ay expression na maaaring true o false. Ang sumusunod na halimbawa ay gumagamit ng operator na `==`, na nagko-compare ng dalawang operands at gumagawa ng `True` kung sila ay equal at `False` kung hindi:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

Ang `True` at `False` ay espesyal na values na kabilang sa class na `bool`; hindi sila strings:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Ang `==` operator ay isa sa mga *comparison operators*; ang iba ay:

```
x != y               # x is not equal to y
x > y                # x is greater than y
x < y                # x is less than y
x >= y               # x is greater than or equal to y
x <= y               # x is less than or equal to y
x is y               # x is the same as y
x is not y           # x is not the same as y
```

Bagaman ang mga operations na ito ay malamang pamilyar sa iyo, ang Python symbols ay iba sa mathematical symbols para sa parehong operations. Karaniwang error ay ang paggamit ng single equal sign (`=`) imbes na double equal sign (`==`). Tandaan na ang `=` ay assignment operator at ang `==` ay comparison operator. Walang ganitong bagay na `=<` o `=>`.

3.2 Logical operators

Mayroong tatlong *logical operators*: `and`, `or`, at `not`. Ang semantics (kahulugan) ng mga operators na ito ay katulad ng kanilang kahulugan sa English. Halimbawa,

```
x > 0 and x < 10
```

ay true lang kung ang `x` ay mas malaki sa 0 *at* mas maliit sa 10.

Ang `n%2 == 0 or n%3 == 0` ay true kung *alinman* sa conditions ay true, iyon ay, kung ang numero ay divisible sa 2 *o* 3.

Sa wakas, ang `not` operator ay nagne-negate ng boolean expression, kaya ang `not (x > y)` ay true kung ang `x > y` ay false.

```
>>> x = 1
>>> y = 2
>>> x > y
False
>>> not (x > y)
True
```

Strictly speaking, ang mga operands ng logical operators ay dapat boolean expressions, pero ang Python ay hindi masyadong strict. Anumang nonzero number ay binibigyang-kahulugan bilang "true."

```
>>> 17 and True
True
```

Ang flexibility na ito ay maaaring kapaki-pakinabang sa ilang situations, pero may ilang subtleties dito na maaaring nakakalito. Maaaring gusto mong iwasan ito hanggang sigurado ka na alam mo ang ginagawa mo.

3.3 Conditional execution

Para sumulat ng kapaki-pakinabang na programs, halos palaging kailangan natin ang kakayahang suriin ang conditions at baguhin ang behavior ng program ayon dito. Ang *Conditional statements* ay nagbibigay sa atin ng kakayahang ito. Ang pinakasimpleng form ay ang `if` statement:

```
if x > 0 :
    print('x is positive')
```

Ang boolean expression pagkatapos ng `if` statement ay tinatawag na *condition*. Tinatapos natin ang `if` statement ng colon character (:) at ang linya/lines pagkatapos ng if statement ay naka-indent.

Kung ang logical condition ay true, pagkatapos ang indented statement ay na-e-execute. Kung ang logical condition ay false, ang indented statement ay na-skip.

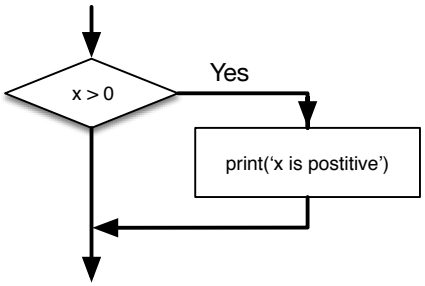

Figure 3.1: If Logic

Ang `if` statements ay may parehong structure sa function definitions o `for` loops[1]. Ang statement ay binubuo ng header line na nagtatapos sa colon character (:) na sinusundan ng indented block. Ang mga statements na ganito ay tinatawag na *compound statements* dahil umaabot sila sa higit sa isang linya.

```
if x > y:
    print(x)
    print(y)
```

Walang limitasyon sa bilang ng statements na maaaring lumabas sa body, pero dapat may hindi bababa sa isa. Paminsan-minsan, kapaki-pakinabang na magkaroon ng body na walang statements (karaniwang bilang place holder para sa code na hindi mo pa nasusulat). Sa kasong iyon, maaari mong gamitin ang `pass` statement para makapasa sa Python interpreter check, na walang ginagawa.

```
if x < 0 :
    pass    # need to handle negative values, do nothing for now.
```

Kung mag-e-enter ka ng `if` statement sa Python interpreter, ang prompt ay magbabago mula sa tatlong chevrons (»>) patungo sa tatlong dots (...) para ipahiwatig na nasa gitna ka ng block ng statements, tulad ng ipinakita sa ibaba:

```
>>> x = 3
>>> if x < 10:
...     print('Small')
...
Small
>>>
```

Kapag gumagamit ng Python interpreter, kailangan mong mag-iwan ng blank line sa dulo ng block, kung hindi ay magre-return ang Python ng error:

```
>>> x = 3
>>> if x < 10:
...     print('Small')
```

[1] Matututunan natin ang functions sa Chapter 4 at loops sa Chapter 5.

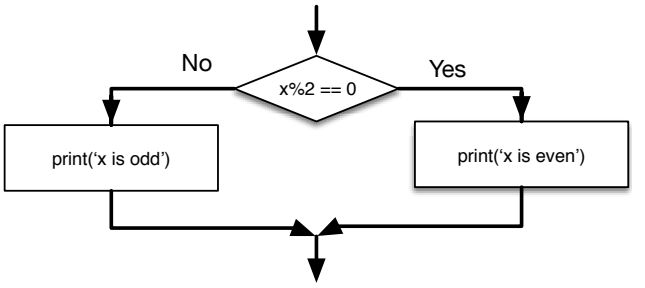

Figure 3.2: If-Then-Else Logic

```
... print('Done')
  File "<stdin>", line 3
    print('Done')
        ^
SyntaxError: invalid syntax
```

Ang blank line sa dulo ng block ng statements ay hindi kailangan kapag sumusulat at nag-e-execute ng script, pero maaari itong mapabuti ang readability ng iyong code.

3.4 Alternative execution

Ang pangalawang form ng `if` statement ay *alternative execution*, kung saan mayroong dalawang posibilidad at ang condition ay tumutukoy kung alin ang na-e-execute. Ang syntax ay ganito:

```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```

Kung ang remainder kapag ang `x` ay hinati sa 2 ay 0, pagkatapos alam natin na ang `x` ay even, at ang program ay nagdi-display ng mensahe tungkol dito. Kung ang condition ay false, ang pangalawang set ng statements ay na-e-execute.

Dahil ang condition ay dapat na true o false, eksaktong isa sa mga alternatives ang ma-e-execute. Ang mga alternatives ay tinatawag na *branches*, dahil sila ay branches sa flow ng execution.

3.5 Chained conditionals

Minsan mayroong higit sa dalawang posibilidad at kailangan natin ng higit sa dalawang branches. Isang paraan para ipahayag ang computation na ganito ay *chained conditional*:

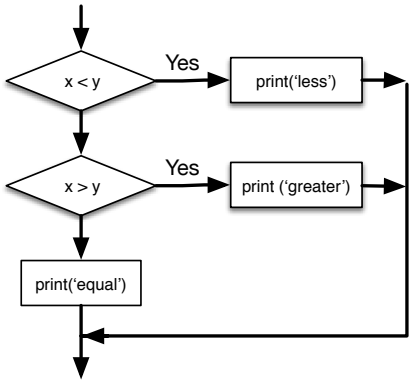

Figure 3.3: If-Then-ElseIf Logic

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

Ang `elif` ay abbreviation ng "else if." Muli, eksaktong isang branch ang ma-e-execute.

Walang limitasyon sa bilang ng `elif` statements. Kung mayroong `else` clause, dapat ito ay nasa dulo, pero hindi kailangan na mayroon.

```
if choice == 'a':
    print('Bad guess')
elif choice == 'b':
    print('Good guess')
elif choice == 'c':
    print('Close, but not correct')
```

Ang bawat condition ay sinusuri sa order. Kung ang una ay false, ang susunod ay sinusuri, at iba pa. Kung ang isa sa kanila ay true, ang corresponding branch ay na-e-execute, at ang statement ay nagtatapos. Kahit na higit sa isang condition ay true, ang unang true branch lang ang na-e-execute.

3.6 Nested conditionals

Ang isang conditional ay maaari ring i-nest sa loob ng isa pa. Maaari nating isulat ang three-branch example na ganito:

```
if x == y:
    print('x and y are equal')
else:
```

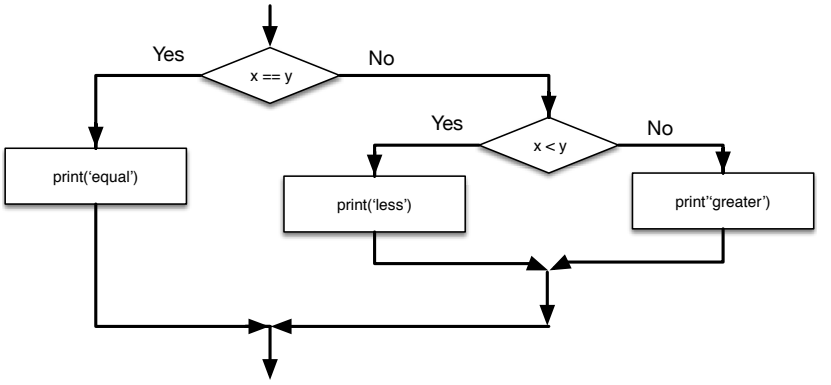

Figure 3.4: Nested If Statements

```
if x < y:
    print('x is less than y')
else:
    print('x is greater than y')
```

Ang outer conditional ay naglalaman ng dalawang branches. Ang unang branch ay naglalaman ng simpleng statement. Ang pangalawang branch ay naglalaman ng iba pang `if` statement, na may dalawang branches din. Ang dalawang branches na iyon ay parehong simpleng statements, bagaman maaari silang maging conditional statements din.

Bagaman ang indentation ng statements ay ginagawang halata ang structure, ang *nested conditionals* ay nagiging mahirap basahin nang napakabilis. Sa pangkalahatan, magandang ideya na iwasan ang mga ito kapag maaari.

Ang logical operators ay kadalasang nagbibigay ng paraan para gawing simple ang nested conditional statements. Halimbawa, maaari nating muling isulat ang sumusunod na code gamit ang isang conditional:

```
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number.')
```

Ang `print` statement ay na-e-execute lang kapag pumasa tayo sa parehong conditionals. Maaari nating makuha ang parehong epekto gamit ang `and` operator:

```
if 0 < x and x < 10:
    print('x is a positive single-digit number.')
```

3.7 Catching exceptions using try and except

Mas maaga nakita natin ang code segment kung saan ginamit natin ang `input` at `int` functions para basahin at i-parse ang integer number na na-enter ng user. Nakita din natin kung gaano kadelikado ang paggawa nito:

```
>>> prompt = "What is the air velocity of an unladen swallow?\n"
>>> speed = input(prompt)
What is the air velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with
base 10: 'What do you mean, an African or a European swallow?'
>>>
```

Kapag nag-e-execute tayo ng mga statements na ito sa Python interpreter, nakakakuha tayo ng bagong prompt mula sa interpreter, iniisip natin ang "oops", at nagpapatuloy sa susunod nating statement.

Gayunpaman kung ilalagay mo ang code na ito sa Python script at mangyari ang error na ito, ang iyong script ay agad na humihinto sa track nito na may traceback. Hindi ito nag-e-execute ng sumusunod na statement.

Narito ang sample program para i-convert ang Fahrenheit temperature sa Celsius temperature:

```
inp = input('Enter Fahrenheit Temperature: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)

# Code: https://www.py4e.com/code3/fahren.py
```

Kung i-e-execute natin ang code na ito at bigyan ito ng invalid input, simpleng nabibigo ito na may unfriendly error message:

```
python fahren.py
Enter Fahrenheit Temperature:72
22.22222222222222

python fahren.py
Enter Fahrenheit Temperature:fred
Traceback (most recent call last):
  File "fahren.py", line 2, in <module>
    fahr = float(inp)
ValueError: could not convert string to float: 'fred'
```

Mayroong conditional execution structure na built-in sa Python para haharapin ang mga uri ng expected at unexpected errors na tinatawag na "try / except". Ang layunin ng `try` at `except` ay alam mo na ang ilang sequence ng instruction(s) ay maaaring may problema at gusto mong magdagdag ng ilang statements para i-execute kung may error na mangyari. Ang mga extra statements na ito (ang except block) ay hindi pinapansin kung walang error.

Maaari mong isipin ang `try` at `except` feature sa Python bilang "insurance policy" sa sequence ng statements.

Maaari nating muling isulat ang temperature converter natin tulad ng sumusunod:

```
inp = input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Please enter a number')

# Code: https://www.py4e.com/code3/fahren2.py
```

Ang Python ay nagsisimula sa pag-e-execute ng sequence ng statements sa `try` block. Kung maayos ang lahat, i-skip nito ang `except` block at magpapatuloy. Kung may exception na mangyari sa `try` block, ang Python ay lumalabas sa `try` block at nag-e-execute ng sequence ng statements sa `except` block.

```
python fahren2.py
Enter Fahrenheit Temperature:72
22.22222222222222

python fahren2.py
Enter Fahrenheit Temperature:fred
Please enter a number
```

Ang pagharap sa exception gamit ang `try` statement ay tinatawag na *catching* ng exception. Sa halimbawang ito, ang `except` clause ay nagpi-print ng error message. Sa pangkalahatan, ang pag-catch ng exception ay nagbibigay sa iyo ng pagkakataon na ayusin ang problema, o subukan ulit, o hindi bababa ay tapusin ang program nang maayos.

3.8 Short-circuit evaluation of logical expressions

Kapag nagpo-process ang Python ng logical expression tulad ng `x >= 2 and (x/y) > 2`, i-e-evaluate nito ang expression mula kaliwa hanggang kanan. Dahil sa definition ng `and`, kung ang `x` ay mas maliit sa 2, ang expression na `x >= 2` ay `False` at kaya ang buong expression ay `False` anuman kung ang `(x/y) > 2` ay nag-e-evaluate sa `True` o `False`.

Kapag nakita ng Python na walang makukuha sa pag-e-evaluate ng natitirang bahagi ng logical expression, humihinto ito sa evaluation at hindi ginagawa ang computations sa natitirang bahagi ng logical expression. Kapag ang evaluation ng logical expression ay humihinto dahil ang overall value ay alam na, ito ay tinatawag na *short-circuiting* ang evaluation.

Habang ito ay maaaring mukhang fine point, ang short-circuit behavior ay nag-dudulot sa matalinong technique na tinatawag na *guardian pattern*. Isaalang-alang ang sumusunod na code sequence sa Python interpreter:

```
>>> x = 6
>>> y = 2
```

```
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Ang pangatlong calculation ay nabigo dahil ang Python ay nag-e-evaluate ng `(x/y)` at ang `y` ay zero, na nagdudulot ng runtime error. Pero ang una at pangalawang halimbawa ay *hindi* nabigo dahil sa unang calculation ang `y` ay non zero at sa pangalawa ang unang parte ng mga expressions na ito na `x >= 2` ay nag-e-evaluate sa `False` kaya ang `(x/y)` ay hindi kailanman na-e-execute dahil sa *short-circuit* rule at walang error.

Maaari nating gawin ang logical expression para strategically maglagay ng *guard* evaluation bago lang ang evaluation na maaaring magdulot ng error tulad ng sumusunod:

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Sa unang logical expression, ang `x >= 2` ay `False` kaya ang evaluation ay humihinto sa `and`. Sa pangalawang logical expression, ang `x >= 2` ay `True` pero ang `y != 0` ay `False` kaya hindi natin naabot ang `(x/y)`.

Sa pangatlong logical expression, ang `y != 0` ay *pagkatapos* ng `(x/y)` calculation kaya ang expression ay nabibigo na may error.

Sa pangalawang expression, sinasabi natin na ang `y != 0` ay nagsisilbing *guard* para masiguro na i-e-execute lang natin ang `(x/y)` kung ang `y` ay non-zero.

3.9 Debugging

Ang traceback na ipinapakita ng Python kapag may error ay naglalaman ng maraming information, pero maaari itong maging overwhelming. Ang pinakakapakipakinabang na parts ay karaniwang:

- Anong uri ng error ito, at
- Saan ito nangyari.

Ang syntax errors ay karaniwang madaling hanapin, pero may ilang gotchas. Ang whitespace errors ay maaaring nakakalito dahil ang spaces at tabs ay invisible at sanay tayong hindi pinapansin ang mga ito.

```
>>> x = 5
>>>  y = 6
  File "<stdin>", line 1
    y = 6
    ^
IndentationError: unexpected indent
```

Sa halimbawang ito, ang problema ay ang pangalawang linya ay naka-indent ng isang space. Pero ang error message ay tumuturo sa `y`, na nakakalito. Sa pangkalahatan, ang error messages ay nagpapahiwatig kung saan natuklasan ang problema, pero ang actual error ay maaaring mas maaga sa code, minsan sa naunang linya.

Sa pangkalahatan, ang error messages ay nagsasabi sa iyo kung saan natuklasan ang problema, pero iyon ay kadalasang hindi kung saan ito naging sanhi.

3.10 Glossary

body Ang sequence ng statements sa loob ng compound statement.
boolean expression Expression na ang value ay maaaring `True` o `False`.
branch Isa sa mga alternative sequences ng statements sa conditional statement.

chained conditional Conditional statement na may serye ng alternative branches.
comparison operator Isa sa mga operators na nagko-compare ng mga operands nito: `==`, `!=`, `>`, `<`, `>=`, at `<=`.
conditional statement Statement na kumokontrol sa flow ng execution depende sa ilang condition.
condition Ang boolean expression sa conditional statement na tumutukoy kung aling branch ang na-e-execute.
compound statement Statement na binubuo ng header at body. Ang header ay nagtatapos sa colon (:). Ang body ay naka-indent relative sa header.
guardian pattern Kung saan gumagawa tayo ng logical expression na may additional comparisons para samantalahin ang short-circuit behavior.
logical operator Isa sa mga operators na nagko-combine ng boolean expressions: `and`, `or`, at `not`.

nested conditional Conditional statement na lumalabas sa isa sa mga branches ng iba pang conditional statement.

traceback List ng mga functions na nag-e-execute, na na-print kapag may exception na nangyari.

short circuit Kapag ang Python ay nasa gitna ng pag-e-evaluate ng logical expression at humihinto sa evaluation dahil alam na ng Python ang final value para sa expression nang hindi kailangan i-evaluate ang natitirang bahagi ng expression.

3.11 Exercises

Exercise 1: Muling isulat ang iyong pay computation para bigyan ang employee ng 1.5 beses ang hourly rate para sa hours na nagtrabaho sa itaas ng 40 hours.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

Exercise 2: Muling isulat ang iyong pay program gamit ang `try` at `except` para ang iyong program ay haharapin ang non-numeric input nang maayos sa pamamagitan ng pag-print ng mensahe at pag-exit sa program. Ang sumusunod ay nagpapakita ng dalawang executions ng program:

```
Enter Hours: 20
Enter Rate: nine
Error, please enter numeric input

Enter Hours: forty
Error, please enter numeric input
```

Exercise 3: Sumulat ng program para mag-prompt para sa score sa pagitan ng 0.0 at 1.0. Kung ang score ay nasa labas ng range, mag-print ng error message. Kung ang score ay sa pagitan ng 0.0 at 1.0, mag-print ng grade gamit ang sumusunod na table:

```
 Score   Grade
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
 < 0.6     F

Enter score: 0.95
A

Enter score: perfect
Bad score
```

```
Enter score: 10.0
Bad score

Enter score: 0.75
C

Enter score: 0.5
F
```

Patakbuhin ang program nang paulit-ulit tulad ng ipinakita sa itaas para i-test ang iba't ibang values para sa input.

Chapter 4

Functions

4.1 Function calls

Sa konteksto ng programming, ang *function* ay pinangalanang sequence ng statements na gumagawa ng computation. Kapag nagde-define ka ng function, tinutukoy mo ang pangalan at ang sequence ng statements. Mamaya, maaari mong "tawagin" ang function sa pamamagitan ng pangalan. Nakita na natin ang isang halimbawa ng *function call*:

```
>>> type(32)
<class 'int'>
```

Ang pangalan ng function ay `type`. Ang expression sa parentheses ay tinatawag na *argument* ng function. Ang argument ay value o variable na ipinapasa natin sa function bilang input sa function. Ang result, para sa `type` function, ay ang type ng argument.

Karaniwang sinasabi na ang function ay "tumatanggap" ng argument at "nagre-return" ng result. Ang result ay tinatawag na *return value*.

4.2 Built-in functions

Ang Python ay nagbibigay ng maraming important built-in functions na maaari nating gamitin nang hindi kailangan magbigay ng function definition. Ang mga creators ng Python ay sumulat ng set ng functions para solusyonan ang common problems at isinama sila sa Python para sa atin na gamitin.

Ang `max` at `min` functions ay nagbibigay sa atin ng pinakamalaki at pinakamaliit na values sa list, ayon sa pagkakabanggit:

```
>>> max('Hello world')
'w'
>>> min('Hello world')
' '
>>>
```

Ang `max` function ay nagsasabi sa atin ng "largest character" sa string (na lumalabas na letrang "w") at ang `min` function ay nagpapakita sa atin ng smallest character (na lumalabas na space).

Ang isa pang napakakaraniwang built-in function ay ang `len` function na nagsasabi sa atin kung ilang items ang nasa argument nito. Kung ang argument sa `len` ay string, nagre-return ito ng bilang ng characters sa string.

```
>>> len('Hello world')
11
>>>
```

Ang mga functions na ito ay hindi limitado sa pagtingin sa strings. Maaari silang gumana sa anumang set ng values, tulad ng makikita natin sa susunod na chapters.

Dapat mong ituring ang mga pangalan ng built-in functions bilang reserved words (i.e., iwasan ang paggamit ng "max" bilang variable name).

4.3 Type conversion functions

Ang Python ay nagbibigay din ng built-in functions na nagko-convert ng values mula sa isang type patungo sa iba. Ang `int` function ay tumatanggap ng anumang value at nagko-convert nito sa integer, kung kaya, o nagre-reklamo kung hindi:

```
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int() with base 10: 'Hello'
```

Ang `int` ay maaaring mag-convert ng floating-point values sa integers, pero hindi ito nagro-round off; tinatanggal nito ang fraction part:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Ang `float` ay nagko-convert ng integers at strings sa floating-point numbers:

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Sa wakas, ang `str` ay nagko-convert ng argument nito sa string:

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

4.4 Math functions

Ang Python ay may `math` module na nagbibigay ng karamihan sa pamilyar na mathematical functions. Bago natin magamit ang module, kailangan nating i-import ito:

```
>>> import math
```

Ang statement na ito ay gumagawa ng *module object* na may pangalang math. Kung i-print mo ang module object, makakakuha ka ng ilang information tungkol dito:

```
>>> print(math)
<module 'math' (built-in)>
```

Ang module object ay naglalaman ng mga functions at variables na tinukoy sa module. Para ma-access ang isa sa mga functions, kailangan mong tukuyin ang pangalan ng module at ang pangalan ng function, na pinaghihiwalay ng dot (kilala din bilang period). Ang format na ito ay tinatawag na *dot notation*.

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)
```

Ang unang halimbawa ay nagko-compute ng logarithm base 10 ng signal-to-noise ratio. Ang math module ay nagbibigay din ng function na tinatawag na `log` na nagko-compute ng logarithms base e.

Ang pangalawang halimbawa ay nakakahanap ng sine ng `radians`. Ang pangalan ng variable ay hint na ang `sin` at ang iba pang trigonometric functions (`cos`, `tan`, etc.) ay tumatanggap ng arguments sa radians. Para i-convert mula sa degrees patungo sa radians, hatiin sa 360 at i-multiply sa 2π:

```
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
0.7071067811865476
```

Ang expression na `math.pi` ay kumukuha ng variable na `pi` mula sa math module. Ang value ng variable na ito ay approximation ng π, tumpak sa humigit-kumulang 15 digits.

Kung alam mo ang trigonometry mo, maaari mong suriin ang naunang result sa pamamagitan ng pag-compare nito sa square root ng dalawa na hinati sa dalawa:

```
>>> math.sqrt(2) / 2.0
0.7071067811865476
```

4.5 Random numbers

Sa parehong inputs, karamihan ng computer programs ay gumagawa ng parehong outputs sa bawat pagkakataon, kaya sinasabi na sila ay *deterministic*. Ang Determinism ay karaniwang magandang bagay, dahil inaasahan natin na ang parehong calculation ay magbibigay ng parehong result. Para sa ilang applications, gayunpaman, gusto natin na ang computer ay unpredictable. Ang mga games ay halatang halimbawa, pero mayroon pa.

Ang paggawa ng program na talagang nondeterministic ay lumalabas na hindi masyadong madali, pero may mga paraan para gawin itong hindi bababa ay mukhang nondeterministic. Isa sa kanila ay gamitin ang *algorithms* na gumagawa ng *pseudorandom* numbers. Ang Pseudorandom numbers ay hindi talagang random dahil ginagawa sila ng deterministic computation, pero sa pagtingin lang sa mga numero ay halos imposibleng makilala sila mula sa random.

Ang `random` module ay nagbibigay ng functions na gumagawa ng pseudorandom numbers (na tatawagin ko lang na "random" mula ngayon).

Ang function na `random` ay nagre-return ng random float sa pagitan ng 0.0 at 1.0 (kasama ang 0.0 pero hindi ang 1.0). Sa bawat pagtawag sa `random`, nakakakuha ka ng susunod na numero sa mahabang serye. Para makita ang sample, patakbuhin ang loop na ito:

```
import random

for i in range(10):
    x = random.random()
    print(x)
```

Ang program na ito ay gumagawa ng sumusunod na list ng 10 random numbers sa pagitan ng 0.0 at hanggang pero hindi kasama ang 1.0.

```
0.11132867921152356
0.5950949227890241
0.04820265884996877
0.841003109276478
0.997914947094958
0.04842330803368111
0.7416295948208405
0.510535245390327
0.27447040171978143
0.028511805472785867
```

Exercise 1: Patakbuhin ang program sa iyong system at tingnan kung anong mga numero ang makukuha mo. Patakbuhin ang program nang higit sa isang beses at tingnan kung anong mga numero ang makukuha mo.

Ang `random` function ay isa lang sa maraming functions na naghahandle ng random numbers. Ang function na `randint` ay tumatanggap ng parameters na `low` at `high`, at nagre-return ng integer sa pagitan ng `low` at `high` (kasama ang pareho).

```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
```

Para pumili ng element mula sa sequence nang random, maaari mong gamitin ang `choice`:

```
>>> t = [1, 2, 3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

Ang `random` module ay nagbibigay din ng functions para gumawa ng random values mula sa continuous distributions kasama ang Gaussian, exponential, gamma, at ilan pa.

4.6 Adding new functions

Hanggang ngayon, gumagamit lang tayo ng mga functions na kasama sa Python, pero maaari ring magdagdag ng bagong functions. Ang *function definition* ay tumutukoy sa pangalan ng bagong function at sa sequence ng statements na na-e-execute kapag tinawag ang function. Kapag nagde-define tayo ng function, maaari nating muling gamitin ang function nang paulit-ulit sa buong program natin.

Narito ang halimbawa:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

Ang `def` ay keyword na nagpapahiwatig na ito ay function definition. Ang pangalan ng function ay `print_lyrics`. Ang mga rules para sa function names ay pareho sa variable names: letra, numero at ilang punctuation marks ay legal, pero ang unang character ay hindi maaaring numero. Hindi mo maaaring gamitin ang keyword bilang pangalan ng function, at dapat mong iwasan na magkaroon ng variable at function na may parehong pangalan.

Ang empty parentheses pagkatapos ng pangalan ay nagpapahiwatig na ang function na ito ay hindi tumatanggap ng anumang arguments. Mamaya ay gagawa tayo ng functions na tumatanggap ng arguments bilang kanilang inputs.

Ang unang linya ng function definition ay tinatawag na *header*; ang natitira ay tinatawag na *body*. Ang header ay dapat magtapos sa colon at ang body ay dapat naka-indent. Ayon sa convention, ang indentation ay palaging apat na spaces. Ang body ay maaaring maglalaman ng anumang bilang ng statements.

Kung magta-type ka ng function definition sa interactive mode, ang interpreter ay nagpi-print ng ellipses (. . .) para ipaalam sa iyo na ang definition ay hindi pa kumpleto:

```
>>> def print_lyrics():
...     print("I'm a lumberjack, and I'm okay.")
...     print('I sleep all night and I work all day.')
...
```

Para tapusin ang function, kailangan mong mag-enter ng empty line (hindi ito kailangan sa script).

Ang pagde-define ng function ay gumagawa ng variable na may parehong pangalan.

```
>>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> print(type(print_lyrics))
<class 'function'>
```

Ang value ng `print_lyrics` ay *function object*, na may type na "function".

Ang syntax para tawagin ang bagong function ay pareho sa built-in functions:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Kapag nagde-define ka na ng function, maaari mong gamitin ito sa loob ng iba pang function. Halimbawa, para ulitin ang naunang refrain, maaari tayong sumulat ng function na tinatawag na `repeat_lyrics`:

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

At pagkatapos tawagin ang `repeat_lyrics`:

```
>>> repeat_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Pero hindi talaga ganun ang kanta.

4.7 Definitions and uses

Pinagsasama ang code fragments mula sa naunang section, ang buong program ay ganito:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')


def repeat_lyrics():
    print_lyrics()
    print_lyrics()

repeat_lyrics()

# Code: https://www.py4e.com/code3/lyrics.py
```

Ang program na ito ay may dalawang function definitions: `print_lyrics` at `repeat_lyrics`. Ang function definitions ay na-e-execute katulad ng iba pang statements, pero ang epekto ay paggawa ng function objects. Ang mga statements sa loob ng function ay hindi na-e-execute hanggang tinawag ang function, at ang function definition ay hindi gumagawa ng output.

Tulad ng maaaring inaasahan mo, kailangan mong gumawa ng function bago mo ma-e-execute ito. Sa ibang salita, ang function definition ay dapat na-e-execute bago ang unang pagkakataon na ito ay tinawag.

Exercise 2: Ilipat ang huling linya ng program na ito sa itaas, para ang function call ay lumabas bago ang definitions. Patakbuhin ang program at tingnan kung anong error message ang makukuha mo.

Exercise 3: Ilipat ang function call pabalik sa ibaba at ilipat ang definition ng `print_lyrics` pagkatapos ng definition ng `repeat_lyrics`. Ano ang mangyayari kapag pinatakbo mo ang program na ito?

4.8 Flow of execution

Para masiguro na ang function ay na-define bago ang unang paggamit nito, kailangan mong malaman ang order kung saan ang statements ay na-e-execute, na tinatawag na *flow of execution*.

Ang execution ay palaging nagsisimula sa unang statement ng program. Ang mga statements ay na-e-execute isa-isa, sa order mula itaas hanggang ibaba.

Ang function *definitions* ay hindi binabago ang flow of execution ng program, pero tandaan na ang mga statements sa loob ng function ay hindi na-e-execute hanggang tinawag ang function.

Ang function call ay parang detour sa flow of execution. Sa halip na pumunta sa susunod na statement, ang flow ay tumatalon sa body ng function, nag-e-execute ng lahat ng statements doon, at pagkatapos ay bumabalik para kunin kung saan ito natigil.

Mukhang simple lang iyon, hanggang maalala mo na ang isang function ay maaaring tumawag sa iba. Habang nasa gitna ng isang function, ang program ay maaaring kailangan i-execute ang mga statements sa iba pang function. Pero habang

nag-e-execute ng bagong function na iyon, ang program ay maaaring kailangan i-execute pa ang isa pang function!

Sa kabutihang-palad, ang Python ay magaling sa pagsubaybay kung nasaan ito, kaya sa bawat pagkakataon na natatapos ang function, ang program ay kumukuha kung saan ito natigil sa function na tumawag dito. Kapag nakarating na sa dulo ng program, ito ay nagtatapos.

Ano ang moral ng sordid tale na ito? Kapag nagbabasa ka ng program, hindi mo palaging gusto na basahin mula itaas hanggang ibaba. Minsan mas makatuwiran kung sinusundan mo ang flow of execution.

4.9 Parameters and arguments

Ang ilan sa mga built-in functions na nakita natin ay nangangailangan ng arguments. Halimbawa, kapag tinatawag mo ang `math.sin` ay ipinapasa mo ang numero bilang argument. Ang ilang functions ay tumatanggap ng higit sa isang argument: ang `math.pow` ay tumatanggap ng dalawa, ang base at ang exponent.

Sa loob ng function, ang mga arguments ay na-a-assign sa variables na tinatawag na *parameters*. Narito ang halimbawa ng user-defined function na tumatanggap ng argument:

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

Ang function na ito ay nag-a-assign ng argument sa parameter na may pangalang `bruce`. Kapag tinawag ang function, nagpi-print ito ng value ng parameter (anuman ito) nang dalawang beses.

Ang function na ito ay gumagana sa anumang value na maaaring i-print.

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(17)
17
17
>>> import math
>>> print_twice(math.pi)
3.141592653589793
3.141592653589793
```

Ang parehong rules ng composition na nalalapat sa built-in functions ay nalalapat din sa user-defined functions, kaya maaari nating gamitin ang anumang uri ng expression bilang argument para sa `print_twice`:

```
>>> print_twice('Spam '*4)
Spam Spam Spam Spam
```

```
Spam Spam Spam Spam
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

Ang argument ay na-e-evaluate bago tinawag ang function, kaya sa mga halimbawa ang expressions na `'Spam '*4` at `math.cos(math.pi)` ay na-e-evaluate lang minsan.

Maaari mo ring gamitin ang variable bilang argument:

```
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
Eric, the half a bee.
Eric, the half a bee.
```

Ang pangalan ng variable na ipinapasa natin bilang argument (`michael`) ay walang kinalaman sa pangalan ng parameter (`bruce`). Hindi mahalaga kung ano ang tawag sa value sa pinanggalingan (sa caller); dito sa `print_twice`, tinatawag natin ang lahat na `bruce`.

4.10 Fruitful functions and void functions

Ang ilan sa mga functions na ginagamit natin, tulad ng math functions, ay nagbibigay ng results; dahil walang mas magandang pangalan, tinatawag ko silang *fruitful functions*. Ang iba pang functions, tulad ng `print_twice`, ay gumagawa ng action pero hindi nagre-return ng value. Tinatawag silang *void functions*.

Kapag tumatawag ka ng fruitful function, halos palaging gusto mong gumawa ng isang bagay sa result; halimbawa, maaari mong i-assign ito sa isang variable o gamitin ito bilang parte ng expression:

```
x = math.cos(radians)
golden = (math.sqrt(5) + 1) / 2
```

Kapag tumatawag ka ng function sa interactive mode, ang Python ay nagdi-display ng result:

```
>>> math.sqrt(5)
2.23606797749979
```

Pero sa script, kung tumatawag ka ng fruitful function at hindi mo i-store ang result ng function sa variable, ang return value ay nawawala sa mist!

```
math.sqrt(5)
```

Ang script na ito ay nagko-compute ng square root ng 5, pero dahil hindi nito i-store ang result sa variable o i-display ang result, hindi ito masyadong kapaki-pakinabang.

Ang void functions ay maaaring mag-display ng isang bagay sa screen o may iba pang epekto, pero wala silang return value. Kung susubukan mong i-assign ang result sa variable, makakakuha ka ng espesyal na value na tinatawag na `None`.

```
>>> result = print_twice('Bing')
Bing
Bing
>>> print(result)
None
```

Ang value na `None` ay hindi pareho sa string na "None". Ito ay espesyal na value na may sariling type:

```
>>> print(type(None))
<class 'NoneType'>
```

Para mag-return ng result mula sa function, ginagamit natin ang `return` statement sa function natin. Halimbawa, maaari tayong gumawa ng napakasimpleng function na tinatawag na `addtwo` na nagdadagdag ng dalawang numero at nagre-return ng result.

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)

# Code: https://www.py4e.com/code3/addtwo.py
```

Kapag na-e-execute ang script na ito, ang `print` statement ay magpi-print ng "8" dahil ang `addtwo` function ay tinawag na may 3 at 5 bilang arguments. Sa loob ng function, ang parameters na `a` at `b` ay 3 at 5 ayon sa pagkakabanggit. Ang function ay nagko-compute ng sum ng dalawang numero at inilagay ito sa local function variable na may pangalang `added`. Pagkatapos ginamit nito ang `return` statement para ipadala ang computed value pabalik sa calling code bilang function result, na na-assign sa variable na `x` at na-print.

4.11 Why functions?

Maaaring hindi malinaw kung bakit sulit ang paghihiwalay ng program sa functions. May ilang dahilan:

- Ang paggawa ng bagong function ay nagbibigay sa iyo ng pagkakataon na pangalanan ang grupo ng statements, na ginagawang mas madaling basahin, maintindihan, at i-debug ang program mo.
- Ang functions ay maaaring gawing mas maliit ang program sa pamamagitan ng pag-aalis ng repetitive code. Mamaya, kung gagawa ka ng pagbabago, kailangan mo lang gawin ito sa isang lugar.
- Ang paghahati ng mahabang program sa functions ay nagpapahintulot sa iyo na i-debug ang mga parts isa-isa at pagkatapos ay pagsama-samahin sila sa working whole.
- Ang well-designed functions ay kadalasang kapaki-pakinabang para sa maraming programs. Kapag nasulat at na-debug mo na ang isa, maaari mo na itong muling gamitin.

Sa natitirang bahagi ng libro, kadalasan ay gagamit tayo ng function definition para ipaliwanag ang konsepto. Parte ng skill ng paggawa at paggamit ng functions ay magkaroon ng function na maayos na nakakakuha ng ideya tulad ng "hanapin ang pinakamaliit na value sa list ng values". Mamaya ay ipapakita namin sa iyo ang code na nakakahanap ng pinakamaliit sa list ng values at ipapakita namin ito sa iyo bilang function na may pangalang `min` na tumatanggap ng list ng values bilang argument nito at nagre-return ng pinakamaliit na value sa list.

4.12 Debugging

Kung gumagamit ka ng text editor para sumulat ng iyong scripts, maaari kang makatagpo ng mga problema sa spaces at tabs. Ang pinakamabuting paraan para iwasan ang mga problemang ito ay gamitin ang spaces lang (walang tabs). Karamihan ng text editors na alam ang Python ay ginagawa ito by default, pero ang ilan ay hindi.

Ang tabs at spaces ay karaniwang invisible, na ginagawang mahirap i-debug, kaya subukan mong maghanap ng editor na namamahala ng indentation para sa iyo.

Gayundin, huwag kalimutang i-save ang program mo bago mo ito patakbuhin. Ang ilang development environments ay ginagawa ito nang awtomatiko, pero ang ilan ay hindi. Sa kasong iyon, ang program na tinitingnan mo sa text editor ay hindi pareho sa program na pinatakbo mo.

Ang debugging ay maaaring tumagal ng mahabang panahon kung patuloy mong pinatakbo ang parehong maling program nang paulit-ulit!

Siguraduhin na ang code na tinitingnan mo ay ang code na pinatakbo mo. Kung hindi ka sigurado, maglagay ng isang bagay tulad ng `print("hello")` sa simula ng program at patakbuhin ito ulit. Kung hindi mo makita ang `hello`, hindi mo pinatakbo ang tamang program!

4.13 Glossary

algorithm Pangkalahatang proseso para solusyonan ang kategorya ng mga problema.

argument Value na ibinigay sa function kapag tinawag ang function. Ang value na ito ay na-a-assign sa corresponding parameter sa function.
body Ang sequence ng statements sa loob ng function definition.
composition Paggamit ng expression bilang parte ng mas malaking expression, o statement bilang parte ng mas malaking statement.
deterministic Tungkol sa program na gumagawa ng parehong bagay sa bawat pagtakbo, sa parehong inputs.
dot notation Ang syntax para tawagin ang function sa iba pang module sa pamamagitan ng pagtukoy sa module name na sinusundan ng dot (period) at function name.
flow of execution Ang order kung saan ang statements ay na-e-execute habang tumatakbo ang program.
fruitful function Function na nagre-return ng value.
function Pinangalanang sequence ng statements na gumagawa ng kapaki-pakinabang na operation. Ang mga functions ay maaaring tumanggap o hindi ng arguments at maaaring gumawa o hindi ng result.
function call Statement na nag-e-execute ng function. Binubuo ito ng function name na sinusundan ng argument list.
function definition Statement na gumagawa ng bagong function, na tumutukoy sa pangalan nito, parameters, at mga statements na na-e-execute nito.
function object Value na ginawa ng function definition. Ang pangalan ng function ay variable na tumutukoy sa function object.
header Ang unang linya ng function definition.
import statement Statement na nagbabasa ng module file at gumagawa ng module object.
module object Value na ginawa ng `import` statement na nagbibigay ng access sa data at code na tinukoy sa module.
parameter Pangalan na ginagamit sa loob ng function para tumukoy sa value na ipinasa bilang argument.
pseudorandom Tungkol sa sequence ng mga numero na mukhang random, pero ginawa ng deterministic program.
return value Ang result ng function. Kung ang function call ay ginagamit bilang expression, ang return value ay ang value ng expression.
void function Function na hindi nagre-return ng value.

4.14 Exercises

Exercise 4: Ano ang layunin ng "def" keyword sa Python?

a) Slang ito na nangangahulugang "ang sumusunod na code ay talagang cool"
b) Nagpapahiwatig ito ng simula ng function
c) Nagpapahiwatig ito na ang sumusunod na indented section ng code ay i-store para mamaya
d) Pareho ang b at c ay totoo
e) Wala sa itaas

Exercise 5: Ano ang i-print ng sumusunod na Python program?

```
def fred():
   print("Zap")
```

```
def jane():
   print("ABC")

jane()
fred()
jane()
```

a) Zap ABC jane fred jane
b) Zap ABC Zap
c) ABC Zap jane
d) ABC Zap ABC
e) Zap Zap Zap

Exercise 6: Muling isulat ang iyong pay computation na may time-and-a-half para sa overtime at gumawa ng function na tinatawag na `computepay` na tumatanggap ng dalawang parameters (`hours` at `rate`).

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

Exercise 7: Muling isulat ang grade program mula sa naunang chapter gamit ang function na tinatawag na `computegrade` na tumatanggap ng score bilang parameter nito at nagre-return ng grade bilang string.

```
 Score   Grade
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
 < 0.6     F
```

```
Enter score: 0.95
A

Enter score: perfect
Bad score

Enter score: 10.0
Bad score

Enter score: 0.75
C

Enter score: 0.5
F
```

Patakbuhin ang program nang paulit-ulit para i-test ang iba't ibang values para sa input.

Chapter 5

Iteration

5.1 Updating variables

Karaniwang pattern sa assignment statements ay assignment statement na nag-u-update ng variable, kung saan ang bagong value ng variable ay depende sa luma.

```
x = x + 1
```

Ito ay nangangahulugang "kunin ang kasalukuyang value ng `x`, magdagdag ng 1, at pagkatapos i-update ang `x` gamit ang bagong value."

Kung susubukan mong i-update ang variable na hindi umiiral, makakakuha ka ng error, dahil ang Python ay nag-e-evaluate ng right side bago mag-assign ng value sa `x`:

```
>>> x = x + 1
NameError: name 'x' is not defined
```

Bago mo ma-update ang variable, kailangan mong *i-initialize* ito, karaniwang may simpleng assignment:

```
>>> x = 0
>>> x = x + 1
```

Ang pag-update ng variable sa pamamagitan ng pagdadagdag ng 1 ay tinatawag na *increment*; ang pagbabawas ng 1 ay tinatawag na *decrement*.

5.2 The `while` statement

Ang mga computers ay kadalasang ginagamit para i-automate ang repetitive tasks. Ang pag-uulit ng magkapareho o katulad na tasks nang hindi gumagawa ng errors ay bagay na magaling gawin ng computers at mahirap gawin ng mga tao. Dahil

ang iteration ay napakakaraniwan, ang Python ay nagbibigay ng ilang language features para gawin itong mas madali.

Isang form ng iteration sa Python ay ang `while` statement. Narito ang simpleng program na nagbi-bilang pababa mula lima at pagkatapos nagsasabi ng "Blastoff!".

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

Maaari mong halos basahin ang `while` statement na parang English. Ibig sabihin nito, "Habang ang `n` ay mas malaki sa 0, i-display ang value ng `n` at pagkatapos bawasan ang value ng `n` ng 1. Kapag nakarating ka sa 0, lumabas sa `while` statement at i-display ang salitang `Blastoff!`"

Mas pormal, narito ang flow of execution para sa `while` statement:

1. I-evaluate ang condition, na nagbibigay ng `True` o `False`.
2. Kung ang condition ay false, lumabas sa `while` statement at magpatuloy ng execution sa susunod na statement.
3. Kung ang condition ay true, i-execute ang body at pagkatapos bumalik sa step 1.

Ang uri ng flow na ito ay tinatawag na *loop* dahil ang ikatlong step ay naglo-loop pabalik sa itaas. Tinatawag natin ang bawat pagkakataon na nag-e-execute tayo ng body ng loop bilang *iteration*. Para sa loop sa itaas, sasabihin natin, "Mayroon itong limang iterations", na nangangahulugang ang body ng loop ay na-e-execute ng limang beses.

Ang body ng loop ay dapat baguhin ang value ng isa o higit pang variables para sa huli ang condition ay maging false at ang loop ay magtatapos. Tinatawag natin ang variable na nagbabago sa bawat pagkakataon na nag-e-execute ang loop at kumokontrol kung kailan natatapos ang loop bilang *iteration variable*. Kung walang iteration variable, ang loop ay mag-uulit magpakailanman, na nagreresulta sa *infinite loop*.

5.3 Infinite loops

Walang katapusang pinagmumulan ng kasiyahan para sa mga programmers ay ang obserbasyon na ang mga direksyon sa shampoo, "Lather, rinse, repeat," ay infinite loop dahil walang *iteration variable* na nagsasabi sa iyo kung ilang beses i-execute ang loop.

Sa kaso ng `countdown`, maaari nating patunayan na ang loop ay nagtatapos dahil alam natin na ang value ng `n` ay finite, at nakikita natin na ang value ng `n` ay nagiging mas maliit sa bawat pagkakataon sa loop, kaya sa huli dapat tayong

makarating sa 0. Sa ibang pagkakataon ang loop ay halatang infinite dahil walang iteration variable.

Minsan hindi mo alam na panahon na para tapusin ang loop hanggang makarating ka sa gitna ng body. Sa kasong iyon maaari kang sumulat ng infinite loop nang sadyang at pagkatapos gamitin ang `break` statement para tumalon palabas ng loop.

Ang loop na ito ay halatang *infinite loop* dahil ang logical expression sa `while` statement ay simpleng logical constant na `True`:

```
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Done!')
```

Kung magkakamali ka at patakbuhin ang code na ito, matututunan mo agad kung paano itigil ang runaway Python process sa iyong system o hanapin kung nasaan ang power-off button sa iyong computer. Ang program na ito ay tatakbo magpakailanman o hanggang ang iyong battery ay maubos dahil ang logical expression sa itaas ng loop ay palaging true dahil sa katotohanan na ang expression ay constant value na `True`.

Habang ito ay dysfunctional infinite loop, maaari pa rin nating gamitin ang pattern na ito para gumawa ng kapaki-pakinabang na loops basta maingat nating idagdag ang code sa body ng loop para tahasang lumabas sa loop gamit ang `break` kapag naabot na natin ang exit condition.

Halimbawa, ipagpalagay na gusto mong kumuha ng input mula sa user hanggang mag-type sila ng `done`. Maaari mong isulat:

```
while True:
    line = input('> ')
    if line == 'done':
        break
    print(line)
print('Done!')

# Code: https://www.py4e.com/code3/copytildone1.py
```

Ang loop condition ay `True`, na palaging true, kaya ang loop ay tumatakbo nang paulit-ulit hanggang maabot ang break statement.

Sa bawat pagkakataon, nagpo-prompt ito sa user na may angle bracket. Kung ang user ay nagta-type ng `done`, ang `break` statement ay lumalabas sa loop. Kung hindi, ang program ay nag-e-echo ng anuman ang na-type ng user at bumabalik sa itaas ng loop. Narito ang sample run:

```
> hello there
hello there
> finished
```

```
finished
> done
Done!
```

Ang paraan ng pagsusulat ng `while` loops na ito ay karaniwan dahil maaari mong suriin ang condition kahit saan sa loop (hindi lang sa itaas) at maaari mong ipahayag ang stop condition nang positibo ("huminto kapag nangyari ito") imbes na negatibo ("magpatuloy hanggang mangyari iyon.").

5.4 Finishing iterations with `continue`

Minsan nasa iteration ka ng loop at gusto mong tapusin ang kasalukuyang iteration at agad na tumalon sa susunod na iteration. Sa kasong iyon maaari mong gamitin ang `continue` statement para lumaktaw sa susunod na iteration nang hindi tinatapos ang body ng loop para sa kasalukuyang iteration.

Narito ang halimbawa ng loop na kumokopya ng input nito hanggang mag-type ang user ng "done", pero tinatrato ang mga linya na nagsisimula sa hash character bilang mga linya na hindi i-print (parang Python comments).

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')

# Code: https://www.py4e.com/code3/copytildone2.py
```

Narito ang sample run ng bagong program na ito na may `continue` na idinagdag.

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

Lahat ng mga linya ay na-print maliban sa isa na nagsisimula sa hash sign dahil kapag na-e-execute ang `continue`, tinatapos nito ang kasalukuyang iteration at tumatalon pabalik sa `while` statement para simulan ang susunod na iteration, kaya na-skip ang `print` statement.

5.5 Definite loops using for

Minsan gusto nating mag-loop sa *set* ng mga bagay tulad ng list ng mga salita, ang mga linya sa file, o list ng mga numero. Kapag mayroon tayong list ng mga bagay na lalapitan ng loop, maaari tayong gumawa ng *definite* loop gamit ang `for` statement. Tinatawag natin ang `while` statement bilang *indefinite* loop dahil simpleng naglo-loop ito hanggang ang ilang condition ay maging `False`, samantalang ang `for` loop ay naglo-loop sa kilalang set ng items kaya tumatakbo ito sa kasing dami ng iterations na may items sa set.

Ang syntax ng `for` loop ay katulad ng `while` loop na mayroong `for` statement at loop body:

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print('Happy New Year:', friend)
print('Done!')
```

Sa mga termino ng Python, ang variable na `friends` ay list[1] ng tatlong strings at ang `for` loop ay dumadaan sa list at nag-e-execute ng body minsan para sa bawat isa sa tatlong strings sa list na nagreresulta sa output na ito:

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

Ang pag-translate ng `for` loop na ito sa English ay hindi kasing direkta ng `while`, pero kung iisipin mo ang friends bilang *set*, ganito: "Patakbuhin ang statements sa body ng for loop minsan para sa bawat friend *sa* set na may pangalang friends."

Sa pagtingin sa `for` loop, ang *for* at *in* ay reserved Python keywords, at ang `friend` at `friends` ay variables.

```
for friend in friends:
    print('Happy New Year:', friend)
```

Sa partikular, ang `friend` ay ang *iteration variable* para sa for loop. Ang variable na `friend` ay nagbabago para sa bawat iteration ng loop at kumokontrol kung kailan natatapos ang `for` loop. Ang *iteration variable* ay sunud-sunod na dumadaan sa tatlong strings na naka-store sa variable na `friends`.

5.6 Loop patterns

Kadalasan gumagamit tayo ng `for` o `while` loop para dumaan sa list ng items o contents ng file at naghahanap tayo ng isang bagay tulad ng pinakamalaki o pinakamaliit na value ng data na sinasala natin.

Ang mga loops na ito ay karaniwang ginagawa sa pamamagitan ng:

[1] Susuriin natin ang lists nang mas detalyado sa susunod na chapter.

- Pag-i-initialize ng isa o higit pang variables bago magsimula ang loop
- Paggawa ng ilang computation sa bawat item sa loop body, maaaring binabago ang variables sa body ng loop
- Pagtingin sa resulting variables kapag natapos ang loop

Gagamitin natin ang list ng mga numero para ipakita ang concepts at construction ng mga loop patterns na ito.

5.6.1 Counting and summing loops

Halimbawa, para bilangin ang bilang ng items sa list, susulat tayo ng sumusunod na `for` loop:

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)
```

Ini-set natin ang variable na `count` sa zero bago magsimula ang loop, pagkatapos sumusulat tayo ng `for` loop para tumakbo sa list ng mga numero. Ang *iteration* variable natin ay may pangalang `itervar` at habang hindi natin ginagamit ang `itervar` sa loop, kinokontrol nito ang loop at nagdudulot na ang loop body ay ma-e-execute minsan para sa bawat isa sa values sa list.

Sa body ng loop, nagdadagdag tayo ng 1 sa kasalukuyang value ng `count` para sa bawat isa sa values sa list. Habang ang loop ay nag-e-execute, ang value ng `count` ay ang bilang ng values na nakita natin "hanggang ngayon".

Kapag natapos ang loop, ang value ng `count` ay ang kabuuang bilang ng items. Ang kabuuang bilang ay "nahuhulog sa kandungan natin" sa dulo ng loop. Ginagawa natin ang loop para mayroon tayong gusto natin kapag natatapos ang loop.

Ang isa pang katulad na loop na nagko-compute ng kabuuan ng set ng mga numero ay ang sumusunod:

```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Total: ', total)
```

Sa loop na ito ay *ginagamit* natin ang *iteration variable*. Sa halip na simpleng magdagdag ng isa sa `count` tulad sa naunang loop, idinadagdag natin ang actual number (3, 41, 12, etc.) sa running total habang bawat loop iteration. Kung iisipin mo ang variable na `total`, naglalaman ito ng "running total ng values hanggang ngayon". Kaya bago magsimula ang loop ang `total` ay zero dahil hindi pa natin nakikita ang anumang values, habang ang loop ay tumatakbo ang `total` ay ang running total, at sa dulo ng loop ang `total` ay ang overall total ng lahat ng values sa list.

Habang ang loop ay nag-e-execute, ang `total` ay nag-a-accumulate ng sum ng mga elements; ang variable na ginagamit sa ganitong paraan ay minsan tinatawag na *accumulator*.

Ang counting loop o summing loop ay hindi partikular na kapaki-pakinabang sa practice dahil mayroong built-in functions na `len()` at `sum()` na nagko-compute ng bilang ng items sa list at ang kabuuan ng items sa list ayon sa pagkakabanggit.

5.6.2 Maximum and minimum loops

Para hanapin ang pinakamalaking value sa list o sequence, gumagawa tayo ng sumusunod na loop:

```
largest = None
print('Before:', largest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if largest is None or itervar > largest :
        largest = itervar
    print('Loop:', itervar, largest)
print('Largest:', largest)
```

Kapag na-e-execute ang program, ang output ay ganito:

```
Before: None
Loop: 3 3
Loop: 41 41
Loop: 12 41
Loop: 9 41
Loop: 74 74
Loop: 15 74
Largest: 74
```

Ang variable na `largest` ay pinakamabuting iisipin bilang "pinakamalaking value na nakita natin hanggang ngayon". Bago ang loop, ini-set natin ang `largest` sa constant na `None`. Ang `None` ay espesyal na constant value na maaari nating i-store sa variable para markahan ang variable bilang "empty".

Bago magsimula ang loop, ang pinakamalaking value na nakita natin hanggang ngayon ay `None` dahil hindi pa natin nakikita ang anumang values. Habang ang loop ay nag-e-execute, kung ang `largest` ay `None` pagkatapos kinukuha natin ang unang value na nakikita natin bilang pinakamalaki hanggang ngayon. Makikita mo sa unang iteration kapag ang value ng `itervar` ay 3, dahil ang `largest` ay `None`, agad naming ini-set ang `largest` na maging 3.

Pagkatapos ng unang iteration, ang `largest` ay hindi na `None`, kaya ang pangalawang parte ng compound logical expression na sumusuri sa `itervar > largest` ay nagti-trigger lang kapag nakikita natin ang value na mas malaki kaysa sa "pinakamalaki hanggang ngayon". Kapag nakikita natin ang bagong "mas malaki pa" na value kinukuha natin ang bagong value na iyon para sa `largest`. Makikita mo sa program output na ang `largest` ay umuunlad mula 3 patungo 41 patungo 74.

Sa dulo ng loop, nasala na natin ang lahat ng values at ang variable na `largest` ngayon ay naglalaman ng pinakamalaking value sa list.

Para i-compute ang pinakamaliit na numero, ang code ay napakatulad na may isang maliit na pagbabago:

```
smallest = None
print('Before:', smallest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if smallest is None or itervar < smallest:
        smallest = itervar
    print('Loop:', itervar, smallest)
print('Smallest:', smallest)
```

Muli, ang `smallest` ay ang "pinakamaliit hanggang ngayon" bago, habang, at pagkatapos na-e-execute ang loop. Kapag natapos ang loop, ang `smallest` ay naglalaman ng minimum value sa list.

Muli tulad sa counting at summing, ang built-in functions na `max()` at `min()` ay ginagawang hindi kailangan ang pagsusulat ng eksaktong loops na ito.

Ang sumusunod ay simpleng bersyon ng Python built-in `min()` function:

```
def min(values):
    smallest = None
    for value in values:
        if smallest is None or value < smallest:
            smallest = value
    return smallest
```

Sa function version ng smallest code, tinanggal namin ang lahat ng `print` statements para maging katumbas ng `min` function na built-in na sa Python.

5.7 Debugging

Habang nagsisimula kang sumulat ng mas malalaking programs, maaari mong makita na gumugugol ka ng mas maraming panahon sa debugging. Mas maraming code ay nangangahulugang mas maraming pagkakataon na magkamali at mas maraming lugar para magtago ang bugs.

Isang paraan para bawasan ang debugging time mo ay "debugging by bisection." Halimbawa, kung mayroong 100 lines sa program mo at sinusuri mo sila isa-isa, aabutin ng 100 steps.

Sa halip, subukan mong hatiin ang problema sa kalahati. Tingnan ang gitna ng program, o malapit dito, para sa intermediate value na maaari mong suriin. Magdagdag ng `print` statement (o iba pang bagay na may verifiable effect) at patakbuhin ang program.

Kung ang mid-point check ay hindi tama, ang problema ay dapat nasa unang kalahati ng program. Kung tama ito, ang problema ay nasa pangalawang kalahati.

Sa bawat pagkakataon na gumagawa ka ng check na ganito, hinahati mo ang bilang ng lines na kailangan mong hanapin. Pagkatapos ng anim na steps (na mas kaunti kaysa sa 100), makakarating ka sa isa o dalawang linya ng code, hindi bababa sa teorya.

Sa practice hindi palaging malinaw kung ano ang "gitna ng program" at hindi palaging posible na suriin ito. Hindi makatuwiran na bilangin ang mga linya at hanapin ang eksaktong midpoint. Sa halip, isipin ang mga lugar sa program kung saan maaaring may errors at mga lugar kung saan madaling maglagay ng check. Pagkatapos pumili ng lugar kung saan sa tingin mo ang mga pagkakataon ay halos pareho na ang bug ay bago o pagkatapos ng check.

5.8 Glossary

accumulator Variable na ginagamit sa loop para magdagdag o mag-accumulate ng result.

counter Variable na ginagamit sa loop para bilangin ang bilang ng beses na nangyari ang isang bagay. Ini-initialize natin ang counter sa zero at pagkatapos i-increment ang counter sa bawat pagkakataon na gusto nating "bilangin" ang isang bagay.

decrement Update na nagpapababa ng value ng variable.

initialize Assignment na nagbibigay ng initial value sa variable na i-u-update.

increment Update na nagpapataas ng value ng variable (kadalasan ng isa).

infinite loop Loop kung saan ang terminating condition ay hindi kailanman nasiyahan o kung saan walang terminating condition.

iteration Paulit-ulit na execution ng set ng statements gamit ang function na tumatawag sa sarili o loop.

5.9 Exercises

Exercise 1: Sumulat ng program na paulit-ulit na nagbabasa ng integers hanggang ang user ay mag-enter ng "done". Kapag na-enter na ang "done", mag-print ng total, count, at average ng integers. Kung ang user ay mag-enter ng anuman maliban sa integer, tuklasin ang kanilang pagkakamali gamit ang `try` at `except` at mag-print ng error message at lumaktaw sa susunod na integers.

```
Enter a number: 4
Enter a number: 5
Enter a number: bad data
Invalid input
Enter a number: 7
Enter a number: done
16 3 5.333333333333333
```

Exercise 2: Sumulat ng isa pang program na nagpo-prompt para sa list ng mga numero tulad ng sa itaas at sa dulo ay magpi-print ng parehong maximum at minimum ng mga numero imbes na ang average.

Chapter 6

Strings

6.1 A string is a sequence

Ang string ay *sequence* ng characters. Maaari mong ma-access ang mga characters isa-isa gamit ang bracket operator:

```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

Ang pangalawang statement ay kumukuha ng character sa index position 1 mula sa variable na `fruit` at nag-a-assign nito sa variable na `letter`.

Ang expression sa brackets ay tinatawag na *index*. Ang index ay nagpapahiwatig kung aling character sa sequence ang gusto mo (kaya ang pangalan).

Pero maaaring hindi mo makuha ang inaasahan mo:

```
>>> print(letter)
a
```

Para sa karamihan ng mga tao, ang unang letra ng "banana" ay "b", hindi "a". Pero sa Python, ang index ay offset mula sa simula ng string, at ang offset ng unang letra ay zero.

```
>>> letter = fruit[0]
>>> print(letter)
b
```

Kaya ang "b" ay ang 0th letra ("zero-th") ng "banana", ang "a" ay ang 1th letra ("one-th"), at ang "n" ay ang 2th ("two-th") letra.

Maaari mong gamitin ang anumang expression, kasama ang variables at operators, bilang index, pero ang value ng index ay dapat integer. Kung hindi makakakuha ka ng:

```
>>> letter = fruit[1.5]
TypeError: string indices must be integers
```

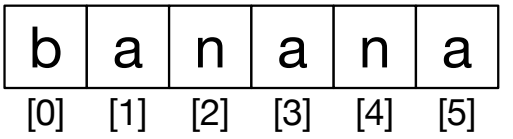

Figure 6.1: String Indexes

6.2 Getting the length of a string using `len`

Ang `len` ay built-in function na nagre-return ng bilang ng characters sa string:

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

Para makuha ang huling letra ng string, maaari kang matukso na subukan ang isang bagay na ganito:

```
>>> length = len(fruit)
>>> last = fruit[length]
IndexError: string index out of range
```

Ang dahilan ng `IndexError` ay walang letra sa "banana" na may index 6. Dahil nagsimula tayo sa pagbilang sa zero, ang anim na letra ay may numero 0 hanggang 5. Para makuha ang huling character, kailangan mong ibawas ang 1 mula sa `length`:

```
>>> last = fruit[length-1]
>>> print(last)
a
```

Bilang alternatibo, maaari mong gamitin ang negative indices, na nagbi-bilang pabalik mula sa dulo ng string. Ang expression na `fruit[-1]` ay nagbibigay ng huling letra, ang `fruit[-2]` ay nagbibigay ng pangalawang huli, at iba pa.

6.3 Traversal through a string with a loop

Maraming computations ang nagsasangkot ng pagproseso ng string isang character sa isang panahon. Kadalasan nagsisimula sila sa simula, pumipili ng bawat character nang sunud-sunod, gumagawa ng isang bagay dito, at nagpapatuloy hanggang sa dulo. Ang pattern ng processing na ito ay tinatawag na *traversal*. Isang paraan para sumulat ng traversal ay gamit ang `while` loop:

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

Ang loop na ito ay dumadaan sa string at nagdi-display ng bawat letra sa isang linya na mag-isa. Ang loop condition ay `index < len(fruit)`, kaya kapag ang `index` ay katumbas ng haba ng string, ang condition ay false, at ang body ng loop ay hindi na-e-execute. Ang huling character na na-access ay ang may index na `len(fruit)-1`, na ang huling character sa string.

Exercise 1: Sumulat ng `while` loop na nagsisimula sa huling character sa string at gumagana pabalik patungo sa unang character sa string, na nagpi-print ng bawat letra sa hiwalay na linya, maliban sa pabalik.

Ang isa pang paraan para sumulat ng traversal ay gamit ang `for` loop:

```
for char in fruit:
    print(char)
```

Sa bawat pagkakataon sa loop, ang susunod na character sa string ay na-a-assign sa variable na `char`. Ang loop ay nagpapatuloy hanggang walang characters na natitira.

6.4 String slices

Ang segment ng string ay tinatawag na *slice*. Ang pagpili ng slice ay katulad ng pagpili ng character:

```
>>> s = 'Monty Python'
>>> print(s[0:5])
Monty
>>> print(s[6:12])
Python
```

Ang operator [n:m] ay nagre-return ng parte ng string mula sa "n-th" character hanggang sa "m-th" character, kasama ang una pero hindi kasama ang huli.

Kung tatanggalin mo ang unang index (bago ang colon), ang slice ay nagsisimula sa simula ng string. Kung tatanggalin mo ang pangalawang index, ang slice ay hanggang sa dulo ng string:

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

Kung ang unang index ay mas malaki o katumbas ng pangalawa ang result ay *empty string*, na kinakatawan ng dalawang quotation marks:

```
>>> fruit = 'banana'
>>> fruit[3:3]
''
```

Ang empty string ay walang characters at may haba na 0, pero bukod sa iyon, pareho ito sa anumang iba pang string.

Exercise 2: Given na ang `fruit` ay string, ano ang ibig sabihin ng `fruit[:]`?

6.5 Strings are immutable

Nakakatukso na gamitin ang operator sa kaliwang bahagi ng assignment, na may layunin na baguhin ang character sa string. Halimbawa:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

Ang "object" sa kasong ito ay ang string at ang "item" ay ang character na sinubukan mong i-assign. Sa ngayon, ang *object* ay pareho sa value, pero pipino natin ang definition na iyon mamaya. Ang *item* ay isa sa mga values sa sequence.

Ang dahilan ng error ay ang strings ay *immutable*, na nangangahulugang hindi mo maaaring baguhin ang existing string. Ang pinakamabuting magagawa mo ay gumawa ng bagong string na variation ng original:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting)
Jello, world!
```

Ang halimbawang ito ay nagko-concatenate ng bagong unang letra sa slice ng `greeting`. Walang epekto ito sa original string.

6.6 Looping and counting

Ang sumusunod na program ay nagbi-bilang ng bilang ng beses na ang letra "a" ay lumalabas sa string:

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```

Ang program na ito ay nagpapakita ng isa pang pattern ng computation na tinatawag na *counter*. Ang variable na `count` ay ini-initialize sa 0 at pagkatapos i-increment sa bawat pagkakataon na makikita ang "a". Kapag ang loop ay lumabas, ang `count` ay naglalaman ng result: ang kabuuang bilang ng a's.

Exercise 3: I-encapsulate ang code na ito sa function na may pangalang `count`, at gawing general para tumanggap ng string at letra bilang arguments.

6.7 The in operator

Ang salitang `in` ay boolean operator na tumatanggap ng dalawang strings at nagre-return ng `True` kung ang una ay lumalabas bilang substring sa pangalawa:

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

6.8 String comparison

Ang comparison operators ay gumagana sa strings. Para makita kung ang dalawang strings ay equal:

```
if word == 'banana':
    print('All right, bananas.')
```

Ang iba pang comparison operations ay kapaki-pakinabang para ilagay ang mga salita sa alphabetical order:

```
if word < 'banana':
    print('Your word,' + word + ', comes before banana.')
elif word > 'banana':
    print('Your word,' + word + ', comes after banana.')
else:
    print('All right, bananas.')
```

Ang Python ay hindi nagha-handle ng uppercase at lowercase letters sa parehong paraan na ginagawa ng mga tao. Lahat ng uppercase letters ay nauuna sa lahat ng lowercase letters, kaya:

```
Your word, Pineapple, comes before banana.
```

Karaniwang paraan para solusyonan ang problemang ito ay i-convert ang strings sa standard format, tulad ng lahat lowercase, bago gawin ang comparison. Tandaan iyon kung sakaling kailangan mong ipagtanggol ang sarili mo laban sa taong armado ng Pineapple.

6.9 String methods

Ang Strings ay halimbawa ng Python *objects*. Ang object ay naglalaman ng parehong data (ang actual string mismo) at *methods*, na epektibong functions na built sa object at available sa anumang *instance* ng object.

Ang Python ay may function na tinatawag na `dir` na nagli-list ng methods available para sa object. Ang `type` function ay nagpapakita ng type ng object at ang `dir` function ay nagpapakita ng available methods.

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
[... 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(self, /)
    Return a capitalized version of the string.

    More specifically, make the first character have upper
    case and the rest lower case.
>>>
```

Habang ang `dir` function ay nagli-list ng methods, at maaari mong gamitin ang `help` para makakuha ng ilang simpleng documentation sa method, mas mabuting source ng documentation para sa string methods ay

https://docs.python.org/library/stdtypes.html#string-methods.

Ang pagtawag sa *method* ay katulad ng pagtawag sa function (ito ay tumatanggap ng arguments at nagre-return ng value) pero ang syntax ay iba. Tinatawag natin ang method sa pamamagitan ng pag-append ng method name sa variable name gamit ang period bilang delimiter.

Halimbawa, ang method na `upper` ay tumatanggap ng string at nagre-return ng bagong string na may lahat uppercase letters:

Sa halip na function syntax na `upper(word)`, ginagamit nito ang method syntax na `word.upper()`.

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print(new_word)
BANANA
```

Ang form ng dot notation na ito ay tumutukoy sa pangalan ng method, `upper`, at ang pangalan ng string na a-applyan ng method, `word`. Ang empty parentheses ay nagpapahiwatig na ang method na ito ay hindi tumatanggap ng argument.

Ang method call ay tinatawag na *invocation*; sa kasong ito, sasabihin natin na nag-i-invoke tayo ng `upper` sa `word`.

Halimbawa, mayroong string method na may pangalang `find` na naghahanap ng posisyon ng isang string sa loob ng isa pa:

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print(index)
1
```

Sa halimbawang ito, nag-i-invoke tayo ng `find` sa `word` at ipinapasa ang letra na hinahanap natin bilang parameter.

Ang `find` method ay maaaring makahanap ng substrings pati na rin characters:

```
>>> word.find('na')
2
```

Maaari itong tumanggap bilang pangalawang argument ng index kung saan dapat ito magsimula:

```
>>> word.find('na', 3)
4
```

Isang karaniwang gawain ay alisin ang white space (spaces, tabs, o newlines) mula sa simula at dulo ng string gamit ang `strip` method:

```
>>> line = '  Here we go  '
>>> line.strip()
'Here we go'
```

Ang ilang methods tulad ng *startswith* ay nagre-return ng boolean values.

```
>>> line = 'Have a nice day'
>>> line.startswith('Have')
True
>>> line.startswith('h')
False
```

Mapapansin mo na ang `startswith` ay nangangailangan ng case na tumugma, kaya minsan kumukuha tayo ng linya at i-map ito lahat sa lowercase bago gumawa ng anumang checking gamit ang `lower` method.

```
>>> line = 'Have a nice day'
>>> line.startswith('h')
False
>>> line.lower()
'have a nice day'
>>> line.lower().startswith('h')
True
```

Sa huling halimbawa, ang method na `lower` ay tinatawag at pagkatapos ginagamit natin ang `startswith` para makita kung ang resulting lowercase string ay nagsisimula sa letrang "h". Hangga't maingat tayo sa order, maaari tayong gumawa ng maraming method calls sa isang expression.

Exercise 4: Mayroong string method na tinatawag na `count` na katulad ng function sa naunang exercise. Basahin ang documentation ng method na ito sa:

https://docs.python.org/library/stdtypes.html#string-methods

Sumulat ng invocation na nagbi-bilang ng bilang ng beses na ang letrang a ay nangyayari sa "banana".

6.10 Parsing strings

Kadalasan, gusto nating tumingin sa string at makahanap ng substring. Halimbawa kung ipinakita sa atin ang serye ng mga linya na na-format tulad ng sumusunod:

`From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008`

at gusto nating kunin lang ang pangalawang kalahati ng address (i.e., `uct.ac.za`) mula sa bawat linya, maaari nating gawin ito sa pamamagitan ng paggamit ng `find` method at string slicing.

Una, hahanapin natin ang posisyon ng at-sign sa string. Pagkatapos hahanapin natin ang posisyon ng unang space *pagkatapos* ng at-sign. At pagkatapos gagamitin natin ang string slicing para kunin ang parte ng string na hinahanap natin.

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1:sppos]
>>> print(host)
uct.ac.za
>>>
```

Ginagamit natin ang bersyon ng `find` method na nagpapahintulot sa atin na tukuyin ang posisyon sa string kung saan gusto nating magsimulang maghanap ang `find`. Kapag nag-slice tayo, kinukuha natin ang characters mula sa "isa pagkatapos ng at-sign hanggang sa *pero hindi kasama* ang space character".

Ang documentation para sa `find` method ay available sa

https://docs.python.org/library/stdtypes.html#string-methods.

6.11 Formatted String Literals

Ang formatted string literal (kadalasang tinutukoy lang bilang f-string) ay nagpapahintulot na gamitin ang Python expressions sa loob ng string literals. Ito ay nakakamit sa pamamagitan ng pag-prepend ng `f` sa string literal at pag-enclose ng expressions sa curly braces `{}`.

Halimbawa, ang pag-wrap ng variable name sa curly braces sa loob ng f-string ay magdudulot na mapalitan ito ng value nito:

```
>>> camels = 42
>>> f'{camels}'
'42'
```

Ang result ay ang string na '42', na hindi dapat malito sa integer value na 42.

Ang expression ay maaaring lumabas kahit saan sa string, kaya maaari mong i-embed ang value sa sentence:

```
>>> camels = 42
>>> f'I have spotted {camels} camels.'
'I have spotted 42 camels.'
```

Maraming expressions ay maaaring isama sa loob ng isang string literal para gumawa ng mas kumplikadong strings.

```
>>> years = 3
>>> count = .1
>>> species = 'camels'
>>> f'In {years} years I have spotted {count} {species}.'
'In 3 years I have spotted 0.1 camels.'
```

Ang formatted string literals ay makapangyarihan, at maaari silang gumawa ng higit pa sa sakop dito. Maaari kang magbasa pa tungkol sa kanila sa

https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals.

6.12 Debugging

Isang skill na dapat mong linangin habang nagpo-program ay palaging nagtatanong sa sarili, "Ano ang maaaring maging mali dito?" o bilang alternatibo, "Anong baliw na bagay ang maaaring gawin ng user natin para i-crash ang (mukhang) perpektong program natin?"

Halimbawa, tingnan ang program na ginamit natin para ipakita ang `while` loop sa chapter tungkol sa iteration:

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')

# Code: https://www.py4e.com/code3/copytildone2.py
```

Tingnan kung ano ang mangyayari kapag ang user ay nag-e-enter ng empty line ng input:

```
> hello there
hello there
> # don't print this
> print this!
print this!
>
Traceback (most recent call last):
  File "copytildone.py", line 3, in <module>
    if line[0] == '#':
IndexError: string index out of range
```

Ang code ay gumagana nang maayos hanggang ipinakita ang empty line. Pagkatapos walang zero-th character, kaya nakakakuha tayo ng traceback. Mayroong dalawang solusyon sa ito para gawing "safe" ang line three kahit na ang linya ay empty.

Ang isang posibilidad ay simpleng gamitin ang `startswith` method na nagre-return ng `False` kung ang string ay empty.

```
if line.startswith('#'):
```

Ang isa pang paraan ay ligtas na sumulat ng `if` statement gamit ang *guardian* pattern at siguraduhin na ang pangalawang logical expression ay na-e-evaluate lang kung saan may hindi bababa sa isang character sa string:

```
if len(line) > 0 and line[0] == '#':
```

6.13 Glossary

counter Variable na ginagamit para bilangin ang isang bagay, karaniwang ini-initialize sa zero at pagkatapos i-increment.

empty string String na walang characters at may haba na 0, na kinakatawan ng dalawang quotation marks.

flag Boolean variable na ginagamit para ipahiwatig kung ang condition ay true o false.
invocation Statement na tumatawag sa method.
immutable Ang property ng sequence na ang items ay hindi maaaring i-assign.
index Integer value na ginagamit para pumili ng item sa sequence, tulad ng character sa string.
item Isa sa mga values sa sequence.
method Function na nauugnay sa object at tinatawag gamit ang dot notation.
object Isang bagay na maaaring tinukoy ng variable. Sa ngayon, maaari mong gamitin ang "object" at "value" nang magkakapalit.
search Pattern ng traversal na humihinto kapag nakita na ang hinahanap nito.
sequence Ordered set; iyon ay, set ng values kung saan ang bawat value ay nakikilala ng integer index.
slice Parte ng string na tinukoy ng range ng indices.
traverse Mag-iterate sa mga items sa sequence, na gumagawa ng katulad na operation sa bawat isa.

6.14 Exercises

Exercise 5: Slicing strings

Kunin ang sumusunod na Python code na nag-i-store ng string:

```
str = 'X-DSPAM-Confidence: 0.8475'
```

Gamitin ang `find` at string slicing para kunin ang parte ng string pagkatapos ng colon character at pagkatapos gamitin ang `float` function para i-convert ang extracted string sa floating point number.

Exercise 6: String methods

Basahin ang documentation ng string methods sa

https://docs.python.org/library/stdtypes.html#string-methods.

Maaaring gusto mong mag-eksperimento sa ilan sa kanila para masiguro na naiintindihan mo kung paano sila gumagana. Ang `strip` at `replace` ay partikular na kapaki-pakinabang.

Ang documentation ay gumagamit ng syntax na maaaring nakakalito. Halimbawa, sa `find(sub[, start[, end]])`, ang brackets ay nagpapahiwatig ng optional arguments. Kaya ang `sub` ay required, pero ang `start` ay optional, at kung isasama mo ang `start`, pagkatapos ang `end` ay optional.

Chapter 7

Files

7.1 Persistence

Hanggang ngayon, natutunan natin kung paano sumulat ng programs at makipag-ugnayan sa intentions natin sa *Central Processing Unit* gamit ang conditional execution, functions, at iterations. Natutunan natin kung paano gumawa at gumamit ng data structures sa *Main Memory*. Ang CPU at memory ay kung saan gumagana at tumatakbo ang software natin. Ito ay kung saan lahat ng "pag-iisip" ay nangyayari.

Pero kung maaalala mo mula sa hardware architecture discussions natin, kapag ang power ay naka-off, anumang naka-store sa CPU o main memory ay nabubura. Kaya hanggang ngayon, ang mga programs natin ay simpleng transient fun exercises lang para matuto ng Python.

Sa chapter na ito, nagsisimula tayong magtrabaho sa *Secondary Memory* (o files). Ang Secondary memory ay hindi nabubura kapag ang power ay naka-off. O sa kaso ng USB flash drive, ang data na isinusulat natin mula sa programs natin ay maaaring alisin sa system at dalhin sa iba pang system.

Pangunahing tututukan natin ang pagbabasa at pagsusulat ng text files tulad ng mga ginagawa natin sa text editor. Mamaya makikita natin kung paano magtrabaho sa database files na binary files, partikular na idinisenyo para basahin at sulatan sa pamamagitan ng database software.

7.2 Opening files

Kapag gusto nating magbasa o sumulat ng file (sabihin sa hard drive mo), una dapat nating *buksan* ang file. Ang pagbubukas ng file ay nakikipag-ugnayan sa operating system mo, na alam kung saan naka-store ang data para sa bawat file. Kapag nagbubukas ka ng file, hinihiling mo sa operating system na hanapin ang file sa pamamagitan ng pangalan at siguraduhin na umiiral ang file. Sa halimbawang ito, binubuksan natin ang file na *mbox.txt*, na dapat naka-store sa parehong folder kung saan ka kapag nagsimula ka ng Python. Maaari mong i-download ang file na ito mula sa www.py4e.com/code3/mbox.txt

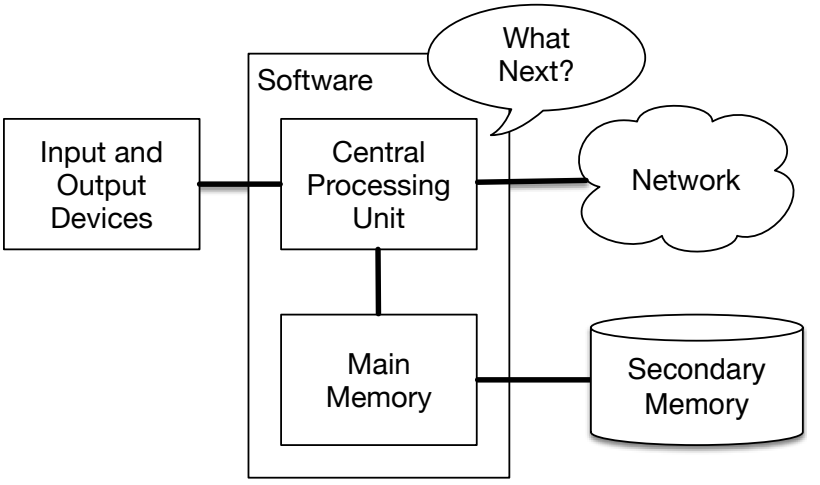

Figure 7.1: Secondary Memory

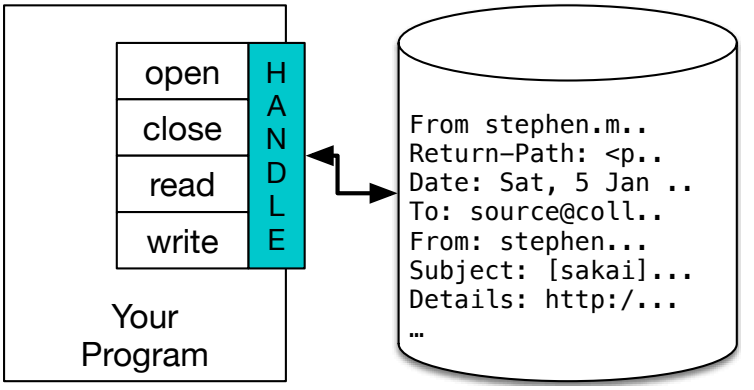

Figure 7.2: A File Handle

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='cp1252'>
```

Kung ang `open` ay matagumpay, ang operating system ay nagre-return sa atin ng *file handle*. Ang file handle ay hindi ang actual data na nasa file, sa halip ito ay "handle" na maaari nating gamitin para basahin ang data. Bibigyan ka ng handle kung ang requested file ay umiiral at mayroon kang tamang permissions para basahin ang file.

Kung ang file ay hindi umiiral, ang `open` ay mabibigo na may traceback at hindi ka makakakuha ng handle para ma-access ang contents ng file:

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'stuff.txt'
```

Mamaya gagamitin natin ang `try` at `except` para harapin nang mas maayos ang situation kung saan sinusubukan nating buksan ang file na hindi umiiral.

7.3 Text files and lines

Ang text file ay maaaring isipin bilang sequence ng mga linya, katulad ng Python string na maaaring isipin bilang sequence ng characters. Halimbawa, ito ay sample ng text file na nagre-record ng mail activity mula sa iba't ibang indibidwal sa open source project development team:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
...
```

Ang buong file ng mail interactions ay available mula sa

www.py4e.com/code3/mbox.txt

at ang pinaikling bersyon ng file ay available mula sa

www.py4e.com/code3/mbox-short.txt

Ang mga files na ito ay nasa standard format para sa file na naglalaman ng maraming mail messages. Ang mga linya na nagsisimula sa "From" ay naghihiwalay ng messages at ang mga linya na nagsisimula sa "From:" ay parte ng messages. Para sa higit pa information tungkol sa mbox format, tingnan ang https://en.wikipedia.org/wiki/Mbox.

Para hatiin ang file sa mga linya, mayroong espesyal na character na kumakatawan sa "end of the line" na tinatawag na *newline* character.

Sa Python, kinakatawan natin ang *newline* character bilang backslash-n sa string constants. Kahit na mukhang dalawang characters ito, ito ay talagang isang character lang. Kapag tinitingnan natin ang variable sa pamamagitan ng pag-enter ng "stuff" sa interpreter, ipinapakita nito sa atin ang `\n` sa string, pero kapag ginagamit natin ang `print` para ipakita ang string, nakikita natin ang string na nahati sa dalawang linya ng newline character.

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
```

```
Y
>>> len(stuff)
3
```

Makikita mo rin na ang haba ng string na `X\nY` ay *tatlo* characters dahil ang newline character ay isang character lang.

Kaya kapag tinitingnan natin ang mga linya sa file, kailangan nating *isipin* na mayroong espesyal na invisible character na tinatawag na newline sa dulo ng bawat linya na nagma-marka ng dulo ng linya.

Kaya ang newline character ay naghihiwalay ng characters sa file sa mga linya.

7.4 Reading files

Habang ang *file handle* ay hindi naglalaman ng data para sa file, napakadaling gumawa ng `for` loop para magbasa at bilangin ang bawat linya sa file:

```
fhand = open('mbox-short.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)

# Code: https://www.py4e.com/code3/open.py
```

Maaari nating gamitin ang file handle bilang sequence sa `for` loop natin. Ang `for` loop natin ay simpleng nagbi-bilang ng bilang ng lines sa file at nagpi-print sa kanila. Ang rough translation ng `for` loop sa English ay, "para sa bawat linya sa file na kinakatawan ng file handle, magdagdag ng isa sa variable na `count`."

Ang dahilan na ang `open` function ay hindi nagbabasa ng buong file ay maaaring napakalaki ang file na may maraming gigabytes ng data. Ang `open` statement ay tumatagal ng parehong dami ng panahon anuman ang laki ng file. Ang `for` loop talaga ang nagdudulot na ang data ay mabasa mula sa file.

Kapag ang file ay binabasa gamit ang `for` loop sa ganitong paraan, ang Python ay nag-aalaga sa paghahati ng data sa file sa hiwalay na mga linya gamit ang newline character. Ang Python ay nagbabasa ng bawat linya hanggang sa newline at kasama ang newline bilang huling character sa variable na `line` para sa bawat iteration ng `for` loop.

Dahil ang `for` loop ay nagbabasa ng data isang linya sa isang panahon, maaari itong mabisa na magbasa at magbilang ng mga linya sa napakalaking files nang hindi naubusan ng main memory para i-store ang data. Ang program sa itaas ay maaaring magbilang ng mga linya sa anumang laking file gamit ang napakakaunting memory dahil ang bawat linya ay binabasa, binibilang, at pagkatapos itinatapon.

Kung alam mo na ang file ay medyo maliit kumpara sa laki ng iyong main memory, maaari mong basahin ang buong file sa isang string gamit ang `read` method sa file handle.

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

Sa halimbawang ito, ang buong contents (lahat ng 94,626 characters) ng file *mbox-short.txt* ay direktang binabasa sa variable na `inp`. Ginagamit natin ang string slicing para i-print ang unang 20 characters ng string data na naka-store sa `inp`.

Kapag ang file ay binabasa sa ganitong paraan, lahat ng characters kasama ang lahat ng mga linya at newline characters ay isang malaking string sa variable na `inp`. Magandang ideya na i-store ang output ng `read` bilang variable dahil ang bawat tawag sa `read` ay nauubos ang resource:

```
>>> fhand = open('mbox-short.txt')
>>> print(len(fhand.read()))
94626
>>> print(len(fhand.read()))
0
```

Tandaan na ang form ng `open` function na ito ay dapat lang gamitin kung ang file data ay magkakasya nang komportable sa main memory ng computer mo. Kung ang file ay masyadong malaki para magkasya sa main memory, dapat mong isulat ang program mo para basahin ang file sa chunks gamit ang `for` o `while` loop.

7.5 Searching through a file

Kapag naghahanap ka sa data sa file, napakakaraniwang pattern na magbasa sa file, hindi pinapansin ang karamihan ng mga linya at nagpo-process lang ng mga linya na tumutugma sa partikular na condition. Maaari nating pagsamahin ang pattern para magbasa ng file kasama ang string methods para gumawa ng simpleng search mechanisms.

Halimbawa, kung gusto nating magbasa ng file at mag-print lang ng mga linya na nagsimula sa prefix na “From:”, maaari nating gamitin ang string method na *startswith* para pumili lang ng mga linya na may gustong prefix:

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:'):
        print(line)

# Code: https://www.py4e.com/code3/search1.py
```

Kapag tumatakbo ang program na ito, nakakakuha tayo ng sumusunod na output:

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

Ang output ay mukhang maganda dahil ang tanging mga linya na nakikita natin ay ang mga nagsisimula sa "From:", pero bakit nakikita natin ang extra blank lines? Ito ay dahil sa invisible na *newline* character na iyon. Ang bawat isa sa mga linya ay nagtatapos sa newline, kaya ang `print` statement ay nagpi-print ng string sa variable na *line* na kasama ang newline at pagkatapos ang `print` ay nagdadagdag ng *isa pa* na newline, na nagreresulta sa double spacing effect na nakikita natin.

Maaari nating gamitin ang line slicing para i-print ang lahat maliban sa huling character, pero ang mas simpleng approach ay gamitin ang *rstrip* method na nagtatanggal ng whitespaces mula sa kanang bahagi ng string tulad ng sumusunod:

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)

# Code: https://www.py4e.com/code3/search2.py
```

Kapag tumatakbo ang program na ito, nakakakuha tayo ng sumusunod na output:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
...
```

Habang ang file processing programs mo ay nagiging mas kumplikado, maaaring gusto mong i-structure ang search loops mo gamit ang `continue`. Ang basic idea ng search loop ay naghahanap ka ng "interesting" lines at epektibong nilalaktawan ang "uninteresting" lines. At pagkatapos kapag nakakita tayo ng interesting line, gumagawa tayo ng isang bagay sa linya na iyon.

Maaari nating i-structure ang loop para sundin ang pattern ng paglaktaw sa uninteresting lines tulad ng sumusunod:

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
```

```
    # Skip 'uninteresting lines'
    if not line.startswith('From:'):
        continue
    # Process our 'interesting' line
    print(line)

# Code: https://www.py4e.com/code3/search3.py
```

Ang output ng program ay pareho. Sa English, ang uninteresting lines ay ang mga hindi nagsisimula sa “From:”, na nilalaktawan natin gamit ang `continue`. Para sa “interesting” lines (i.e., ang mga nagsisimula sa “From:”) ginagawa natin ang processing.

Maaari nating gamitin ang `find` string method para gayahin ang text editor search na nakakahanap ng mga linya kung saan ang search string ay nasa kahit saan sa linya. Dahil ang `find` ay naghahanap ng occurrence ng string sa loob ng ibang string at nagre-return ng posisyon ng string o -1 kung hindi nahanap ang string, maaari tayong sumulat ng sumusunod na loop para ipakita ang mga linya na naglalaman ng string na “@uct.ac.za” (i.e., galing sila sa University of Cape Town sa South Africa):

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1: continue
    print(line)

# Code: https://www.py4e.com/code3/search4.py
```

Na gumagawa ng sumusunod na output:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
...
```

Dito ginagamit din natin ang contracted form ng `if` statement kung saan inilalagay natin ang `continue` sa parehong linya ng `if`. Ang contracted form ng `if` na ito ay gumagana pareho kung ang `continue` ay nasa susunod na linya at naka-indent.

7.6 Letting the user choose the file name

Talagang ayaw nating kailanganin na i-edit ang Python code natin sa bawat pagkakataon na gusto nating i-process ang ibang file. Mas magiging maga-gamit kung hihilingin natin sa user na mag-enter ng file name string sa bawat

pagkakataon na tumatakbo ang program para magamit nila ang program natin sa iba't ibang files nang hindi binabago ang Python code.

Napakasimple gawin ito sa pamamagitan ng pagbasa ng file name mula sa user gamit ang `input` tulad ng sumusunod:

```
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)

# Code: https://www.py4e.com/code3/search6.py
```

Binabasa natin ang file name mula sa user at inilalagay ito sa variable na may pangalang `fname` at binubuksan ang file na iyon. Ngayon maaari na nating patakbuhin ang program nang paulit-ulit sa iba't ibang files.

```
python search6.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search6.py
Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt
```

Bago tumingin sa susunod na section, tingnan ang program sa itaas at tanungin ang sarili mo, "Ano ang maaaring maging mali dito?" o "Ano ang maaaring gawin ng friendly user natin na magdudulot na ang nice little program natin ay lumabas nang hindi maayos na may traceback, na ginagawang hindi masyadong cool tayo sa mga mata ng users natin?"

7.7 Using `try`, `except`, and `open`

Sinabi ko sa iyo na huwag tumingin. Ito na ang huling pagkakataon mo.

Paano kung ang user natin ay nagta-type ng isang bagay na hindi file name?

```
python search6.py
Enter the file name: missing.txt
Traceback (most recent call last):
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'missing.txt'

python search6.py
Enter the file name: na na boo boo
Traceback (most recent call last):
```

```
  File "search6.py", line 2, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'na na boo boo'
```

Huwag tumawa. Ang mga users ay sa huli ay gagawin ang bawat posibleng bagay na magagawa nila para sirain ang programs mo, maaaring hindi sinasadya o may masamang layunin. Bilang matter of fact, mahalagang parte ng anumang software development team ay isang tao o grupo na tinatawag na *Quality Assurance* (o QA para sa maikli) na ang trabaho ay gawin ang pinakamabaliw na bagay na posible sa pagtatangka na sirain ang software na ginawa ng programmer.

Ang QA team ay responsable sa paghahanap ng flaws sa programs bago natin naideliver ang program sa end users na maaaring bumibili ng software o nagbabayad ng suweldo natin para sumulat ng software. Kaya ang QA team ay pinakamabuting kaibigan ng programmer.

Kaya ngayon na nakikita natin ang flaw sa program, maaari nating maayos na ayusin ito gamit ang `try/except` structure. Kailangan nating i-assume na ang `open` call ay maaaring mabigo at magdagdag ng recovery code kapag ang `open` ay nabigo tulad ng sumusunod:

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)

# Code: https://www.py4e.com/code3/search7.py
```

Ang `exit` function ay nagtatapos ng program. Ito ay function na tinatawag natin na hindi kailanman nagre-return. Ngayon kapag ang user natin (o QA team) ay nagta-type ng kalokohan o masamang file names, "hinuhuli" natin sila at nagre-recover nang maayos:

```
python search7.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python search7.py
Enter the file name: na na boo boo
File cannot be opened: na na boo boo
```

Ang pagprotekta sa `open` call ay magandang halimbawa ng tamang paggamit ng `try` at `except` sa Python program. Ginagamit natin ang term na "Pythonic" kapag gumagawa tayo ng isang bagay sa "Python way". Maaari nating sabihin na ang halimbawa sa itaas ay ang Pythonic way para buksan ang file.

Kapag naging mas bihasa ka na sa Python, maaari kang makipag-repartee sa iba pang Python programmers para magpasya kung alin sa dalawang katumbas na solusyon sa problema ang “mas Pythonic”. Ang layunin na maging “mas Pythonic” ay kumukuha ng konsepto na ang programming ay parte engineering at parte art. Hindi tayo palaging interesado lang na gumawa ng isang bagay na gumagana, gusto din natin na ang solusyon natin ay elegant at maa-appreciate bilang elegant ng mga kapantay natin.

7.8 Writing files

Para sumulat ng file, kailangan mong buksan ito na may mode na “w” bilang pangalawang parameter:

```
>>> fout = open('output.txt', 'w')
>>> print(fout)
<_io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252'>
```

Kung ang file ay umiiral na, ang pagbubukas nito sa write mode ay naglilinis ng lumang data at nagsisimula ng fresh, kaya mag-ingat! Kung ang file ay hindi umiiral, bago ang gagawin.

Ang `write` method ng file handle object ay naglalagay ng data sa file, na nagre-return ng bilang ng characters na naisulat. Ang default write mode ay text para sa pagsusulat (at pagbabasa) ng strings.

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
24
```

Muli, ang file object ay nagsu-subaybay kung nasaan ito, kaya kung tatawagin mo ang `write` ulit, idinadagdag nito ang bagong data sa dulo.

Dapat nating siguraduhin na pamahalaan ang dulo ng mga linya habang sumusulat tayo sa file sa pamamagitan ng tahasang paglalagay ng newline character kapag gusto nating tapusin ang linya. Ang `print` statement ay awtomatikong nag-a-append ng newline, pero ang `write` method ay hindi nagdadagdag ng newline nang awtomatiko.

```
>>> line2 = 'the emblem of our land.\n'
>>> fout.write(line2)
24
```

Kapag tapos ka na sa pagsusulat, kailangan mong isara ang file para masiguro na ang huling piraso ng data ay pisikal na naisulat sa disk para hindi ito mawala kung ang power ay mawawala.

```
>>> fout.close()
```

Maaari nating isara ang mga files na binubuksan natin para sa pagbabasa din, pero maaari tayong maging medyo pabaya kung nagbubukas lang tayo ng ilang files dahil sinisiguro ng Python na lahat ng bukas na files ay nagsasara kapag natatapos ang program. Kapag sumusulat tayo ng files, gusto nating tahasang isara ang files para walang maiwan sa pagkakataon.

7.9 Debugging

Kapag nagbabasa at sumusulat ka ng files, maaari kang makatagpo ng mga problema sa whitespace. Ang mga errors na ito ay maaaring mahirap i-debug dahil ang spaces, tabs, at newlines ay karaniwang invisible:

```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2  3
 4
```

Ang built-in function na `repr` ay maaaring makatulong. Tumatanggap ito ng anumang object bilang argument at nagre-return ng string representation ng object. Para sa strings, kinakatawan nito ang whitespace characters gamit ang backslash sequences:

```
>>> print(repr(s))
'1 2\t 3\n 4'
```

Ito ay maaaring makatulong para sa debugging.

Ang isa pang problema na maaaring makatagpo ka ay ang iba't ibang systems ay gumagamit ng iba't ibang characters para ipahiwatig ang dulo ng linya. Ang ilang systems ay gumagamit ng newline, na kinakatawan ng `\n`. Ang iba ay gumagamit ng return character, na kinakatawan ng `\r`. Ang ilan ay gumagamit ng pareho. Kung ililipat mo ang files sa pagitan ng iba't ibang systems, ang mga inconsistencies na ito ay maaaring magdulot ng mga problema.

Para sa karamihan ng systems, mayroong applications para i-convert mula sa isang format patungo sa iba. Maaari mong hanapin ang mga ito (at magbasa pa tungkol sa issue na ito) sa https://www.wikipedia.org/wiki/Newline. O, siyempre, maaari mong isulat ang isa sa iyong sarili.

7.10 Glossary

catch Para pigilan ang exception na tapusin ang program gamit ang `try` at `except` statements.

newline Espesyal na character na ginagamit sa files at strings para ipahiwatig ang dulo ng linya.

Pythonic Technique na gumagana nang elegant sa Python. "Ang paggamit ng try at except ay ang *Pythonic* way para makabawi mula sa missing files".

Quality Assurance Tao o team na nakatuon sa pagtiyak ng overall quality ng software product. Ang QA ay kadalasang kasangkot sa pag-test ng product at pagkilala ng mga problema bago i-release ang product.

text file Sequence ng characters na naka-store sa permanent storage tulad ng hard drive.

7.11 Exercises

Exercise 1: Sumulat ng program para magbasa sa file at mag-print ng contents ng file (linya sa linya) lahat sa upper case. Ang pag-e-execute ng program ay magiging ganito:

```
python shout.py
Enter a file name: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN  5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
     BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
     SAT, 05 JAN 2008 09:14:16 -0500
```

Maaari mong i-download ang file mula sa www.py4e.com/code3/mbox-short.txt

Exercise 2: Sumulat ng program para mag-prompt para sa file name, at pagkatapos magbasa sa file at maghanap ng mga linya ng form:

```
X-DSPAM-Confidence: 0.8475
```

Kapag nakatagpo ka ng linya na nagsisimula sa "X-DSPAM-Confidence:" hatiin ang linya para kunin ang floating-point number sa linya. Bilangin ang mga linyang ito at pagkatapos i-compute ang kabuuan ng spam confidence values mula sa mga linyang ito. Kapag naabot mo na ang dulo ng file, mag-print ng average spam confidence.

```
Enter the file name: mbox.txt
Average spam confidence: 0.894128046745

Enter the file name: mbox-short.txt
Average spam confidence: 0.750718518519
```

I-test ang file mo sa *mbox.txt* at *mbox-short.txt* files.

Exercise 3:

Minsan kapag ang mga programmers ay naiinip o gusto ng kaunting kasiyahan, nagdadagdag sila ng harmless na *Easter Egg* sa program nila. Baguhin ang program na nagpo-prompt sa user para sa file name para mag-print ng nakakatawang mensahe kapag ang user ay nagta-type ng eksaktong file name na "na na boo boo". Ang program ay dapat kumilos nang normal para sa lahat ng iba pang files na umiiral at hindi umiiral. Narito ang sample execution ng program:

```
python egg.py
Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

python egg.py
Enter the file name: missing.tyxt
File cannot be opened: missing.tyxt

python egg.py
Enter the file name: na na boo boo
NA NA BOO BOO TO YOU - You have been punk'd!
```

Hindi namin hinihikayat na maglagay ng Easter Eggs sa programs mo; ito ay exercise lang.

Chapter 8

Lists

8.1 A list is a sequence

Tulad ng string, ang *list* ay sequence ng values. Sa string, ang values ay characters; sa list, maaari silang anumang type. Ang mga values sa lists ay tinatawag na *elements* o minsan *items*.

Mayroong ilang paraan para gumawa ng bagong list; ang pinakasimple ay i-enclose ang elements sa square brackets ("[" at "]"):

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
```

Ang unang halimbawa ay list ng apat na integers. Ang pangalawa ay list ng tatlong strings. Ang elements ng list ay hindi kailangang pareho ang type. Ang sumusunod na list ay naglalaman ng string, float, integer, at (lo!) isa pang list:

```
['spam', 2.0, 5, [10, 20]]
```

Ang list sa loob ng ibang list ay *nested*.

Ang list na walang elements ay tinatawag na empty list; maaari kang gumawa ng isa na may empty brackets, `[]`.

Tulad ng maaaring inaasahan mo, maaari mong i-assign ang list values sa variables:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

8.2 Lists are mutable

Ang syntax para ma-access ang elements ng list ay pareho sa pag-access ng characters ng string: ang bracket operator. Ang expression sa loob ng brackets ay tumutukoy sa index. Tandaan na ang indices ay nagsisimula sa 0:

```
>>> print(cheeses[0])
Cheddar
```

Hindi tulad ng strings, ang lists ay mutable dahil maaari mong baguhin ang order ng items sa list o mag-reassign ng item sa list. Kapag ang bracket operator ay lumalabas sa kaliwang bahagi ng assignment, kinikilala nito ang element ng list na ma-a-assign.

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```

Ang one-th element ng `numbers`, na dati ay 123, ay ngayon 5.

Maaari mong isipin ang list bilang relasyon sa pagitan ng indices at elements. Ang relasyong ito ay tinatawag na *mapping*; ang bawat index ay "nagma-map sa" isa sa mga elements.

Ang list indices ay gumagana sa parehong paraan ng string indices:

- Anumang integer expression ay maaaring gamitin bilang index.
- Kung susubukan mong basahin o sulatan ang element na hindi umiiral, makakakuha ka ng `IndexError`.
- Kung ang index ay may negative value, nagbi-bilang ito pabalik mula sa dulo ng list.

Ang `in` operator ay gumagana din sa lists.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

8.3 Traversing a list

Ang pinakakaraniwang paraan para dumaan sa elements ng list ay gamit ang `for` loop. Ang syntax ay pareho sa strings:

```
for cheese in cheeses:
    print(cheese)
```

Ito ay gumagana nang maayos kung kailangan mo lang basahin ang elements ng list. Pero kung gusto mong sumulat o mag-update ng elements, kailangan mo ang indices. Ang karaniwang paraan para gawin iyon ay pagsamahin ang functions na `range` at `len`:

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

Ang loop na ito ay dumadaan sa list at nag-u-update ng bawat element. Ang `len` ay nagre-return ng bilang ng elements sa list. Ang `range` ay nagre-return ng list ng indices mula 0 hanggang $n - 1$, kung saan ang n ay ang haba ng list. Sa bawat pagkakataon sa loop, ang `i` ay nakakakuha ng index ng susunod na element. Ang assignment statement sa body ay gumagamit ng `i` para basahin ang lumang value ng element at i-assign ang bagong value.

Ang `for` loop sa empty list ay hindi kailanman nag-e-execute ng body:

```
for x in empty:
    print('This never happens.')
```

Bagaman ang list ay maaaring maglalaman ng ibang list, ang nested list ay binibilang pa rin bilang isang element. Ang haba ng list na ito ay apat:

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

8.4 List operations

Ang `+` operator ay nagko-concatenate ng lists:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

Katulad nito, ang `*` operator ay nag-uulit ng list sa ibinigay na bilang ng beses:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Ang unang halimbawa ay nag-uulit ng apat na beses. Ang pangalawang halimbawa ay nag-uulit ng list ng tatlong beses.

8.5 List slices

Ang slice operator ay gumagana din sa lists:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
```

Kung tatanggalin mo ang unang index, ang slice ay nagsisimula sa simula. Kung tatanggalin mo ang pangalawa, ang slice ay hanggang sa dulo. Kaya kung tatanggalin mo ang pareho, ang slice ay kopya ng buong list.

```
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Dahil ang lists ay mutable, kadalasang kapaki-pakinabang na gumawa ng kopya bago gawin ang operations na nagfo-fold, nag-spindle, o nagmu-mutilate ng lists.

Ang slice operator sa kaliwang bahagi ng assignment ay maaaring mag-update ng maraming elements:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print(t)
['a', 'x', 'y', 'd', 'e', 'f']
```

8.6 List methods

Ang Python ay nagbibigay ng methods na gumagana sa lists. Halimbawa, ang `append` ay nagdadagdag ng bagong element sa dulo ng list:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print(t)
['a', 'b', 'c', 'd']
```

Ang `extend` ay tumatanggap ng list bilang argument at nag-a-append ng lahat ng elements:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print(t1)
['a', 'b', 'c', 'd', 'e']
```

Ang halimbawang ito ay nag-iiwan sa `t2` na hindi nabago.

Ang `sort` ay nag-a-arrange ng elements ng list mula mababa hanggang mataas:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print(t)
['a', 'b', 'c', 'd', 'e']
```

Karamihan ng list methods ay void; binabago nila ang list at nagre-return ng `None`. Kung aksidenteng sumulat ka ng `t = t.sort()`, mabibigo ka sa result.

8.7 Deleting elements

Mayroong ilang paraan para tanggalin ang elements mula sa list. Kung alam mo ang index ng element na gusto mo, maaari mong gamitin ang `pop`:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print(t)
['a', 'c']
>>> print(x)
b
```

Ang `pop` ay nagmo-modify ng list at nagre-return ng element na tinanggal. Kung hindi mo ibibigay ang index, tinatanggal at nagre-return ito ng huling element.

Kung hindi mo kailangan ang tinanggal na value, maaari mong gamitin ang `del` statement:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print(t)
['a', 'c']
```

Kung alam mo ang element na gusto mong tanggalin (pero hindi ang index), maaari mong gamitin ang `remove`:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print(t)
['a', 'c']
```

Ang return value mula sa `remove` ay `None`.

Para tanggalin ang higit sa isang element, maaari mong gamitin ang `del` na may slice index:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print(t)
['a', 'f']
```

Tulad ng dati, ang slice ay pumipili ng lahat ng elements hanggang sa, pero hindi kasama, ang pangalawang index.

8.8 Lists and functions

Mayroong ilang built-in functions na maaaring gamitin sa lists na nagpapahintulot sa iyo na mabilis na tumingin sa list nang hindi sumusulat ng sarili mong loops:

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25
```

Ang `sum()` function ay gumagana lang kapag ang list elements ay numbers. Ang iba pang functions (`max()`, `len()`, atbp.) ay gumagana sa lists ng strings at iba pang types na maaaring maihambing.

Maaari nating muling isulat ang naunang program na nag-compute ng average ng list ng numbers na na-enter ng user gamit ang list.

Una, ang program para mag-compute ng average nang walang list:

```
total = 0
count = 0
while (True):
    inp = input('Enter a number: ')
    if inp == 'done': break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print('Average:', average)

# Code: https://www.py4e.com/code3/avenum.py
```

Sa program na ito, mayroon tayong `count` at `total` variables para panatilihin ang bilang at running total ng numbers ng user habang paulit-ulit nating pinaprompt ang user para sa number.

Maaari nating simpleng tandaan ang bawat number habang ini-enter ito ng user at gamitin ang built-in functions para mag-compute ng sum at count sa dulo.

```
numlist = list()
while (True):
    inp = input('Enter a number: ')
    if inp == 'done': break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)

# Code: https://www.py4e.com/code3/avelist.py
```

Gumagawa tayo ng empty list bago magsimula ang loop, at pagkatapos sa bawat pagkakataon na mayroon tayong number, idinadagdag natin ito sa list. Sa dulo ng program, simpleng kinakalkula natin ang sum ng numbers sa list at hinahati ito sa bilang ng numbers sa list para makakuha ng average.

8.9 Lists and strings

Ang string ay sequence ng characters at ang list ay sequence ng values, pero ang list ng characters ay hindi pareho sa string. Para i-convert mula sa string patungo sa list ng characters, maaari mong gamitin ang `list`:

```
>>> s = 'spam'
>>> t = list(s)
>>> print(t)
['s', 'p', 'a', 'm']
```

Dahil ang `list` ay pangalan ng built-in function, dapat mong iwasan ang paggamit nito bilang variable name. Iniiwasan ko rin ang letrang “l” dahil mukhang masyadong katulad ng number na “1”. Kaya iyon ang dahilan kung bakit ginagamit ko ang “t”.

Ang `list` function ay naghahati ng string sa indibidwal na mga letra. Kung gusto mong hatiin ang string sa mga salita, maaari mong gamitin ang `split` method:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print(t)
['pining', 'for', 'the', 'fjords']
>>> print(t[2])
the
```

Kapag ginamit mo na ang `split` para hatiin ang string sa list ng mga salita, maaari mong gamitin ang index operator (square bracket) para tumingin sa partikular na salita sa list.

Maaari mong tawagin ang `split` na may optional argument na tinatawag na *delimiter* na tumutukoy kung aling characters ang gagamitin bilang word boundaries. Ang sumusunod na halimbawa ay gumagamit ng hyphen bilang delimiter:

```
>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> s.split(delimiter)
['spam', 'spam', 'spam']
```

Ang `join` ay kabaligtaran ng `split`. Tumatanggap ito ng list ng strings at nagkoconcatenate ng elements. Ang `join` ay string method, kaya kailangan mong i-invoke ito sa delimiter at ipasa ang list bilang parameter:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> delimiter.join(t)
'pining for the fjords'
```

Sa kasong ito ang delimiter ay space character, kaya ang `join` ay naglalagay ng space sa pagitan ng mga salita. Para mag-concatenate ng strings nang walang spaces, maaari mong gamitin ang empty string, " ", bilang delimiter.

8.10 Parsing lines

Karaniwan kapag nagbabasa tayo ng file gusto nating gumawa ng isang bagay sa mga linya maliban sa simpleng pag-print ng buong linya. Kadalasan gusto nating hanapin ang "interesting lines" at pagkatapos *i-parse* ang linya para hanapin ang ilang interesting na *part* ng linya. Paano kung gusto nating mag-print ng araw ng linggo mula sa mga linyang nagsisimula sa "From"?

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Ang `split` method ay napaka-epektibo kapag nahaharap sa ganitong uri ng problema. Maaari tayong sumulat ng maliit na program na naghahanap ng mga linya kung saan ang linya ay nagsisimula sa "From", `split` ang mga linyang iyon, at pagkatapos mag-print ng ikatlong salita sa linya:

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '): continue
    words = line.split()
    print(words[2])

# Code: https://www.py4e.com/code3/search5.py
```

Figure 8.1: Variables and Objects

Ang program ay gumagawa ng sumusunod na output:

```
Sat
Fri
Fri
Fri
...
```

Mamaya, matututunan natin ang mas sopistikadong techniques para pumili ng mga linya na gagawin at kung paano natin hinahati ang mga linyang iyon para hanapin ang eksaktong piraso ng impormasyon na hinahanap natin.

8.11 Objects and values

Kung e-execute natin ang assignment statements na ito:

```
a = 'banana'
b = 'banana'
```

alam natin na ang `a` at `b` ay parehong tumutukoy sa string, pero hindi natin alam kung tumutukoy sila sa *parehong* string. Mayroong dalawang posibleng estado:

Sa isang kaso, ang `a` at `b` ay tumutukoy sa dalawang magkaibang objects na may parehong value. Sa pangalawang kaso, tumutukoy sila sa parehong object.

Para suriin kung ang dalawang variables ay tumutukoy sa parehong object, maaari mong gamitin ang `is` operator.

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

Sa halimbawang ito, ang Python ay gumawa lang ng isang string object, at pareho ang `a` at `b` ay tumutukoy dito.

Pero kapag gumawa ka ng dalawang lists, makakakuha ka ng dalawang objects:

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

Sa kasong ito sasabihin natin na ang dalawang lists ay *equivalent*, dahil mayroon silang parehong elements, pero hindi *identical*, dahil hindi sila parehong object. Kung ang dalawang objects ay identical, equivalent din sila, pero kung equivalent sila, hindi naman sila kinakailangang identical.

Hanggang ngayon, ginagamit natin ang "object" at "value" nang magkakapalit, pero mas tumpak na sabihin na ang object ay may value. Kung e-execute mo ang `a = [1,2,3]`, ang `a` ay tumutukoy sa list object na ang value ay partikular na sequence ng elements. Kung ang ibang list ay may parehong elements, sasabihin natin na mayroon itong parehong value.

8.12 Aliasing

Kung ang `a` ay tumutukoy sa object at i-assign mo ang `b = a`, pagkatapos parehong variables ay tumutukoy sa parehong object:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

Ang asosasyon ng variable sa object ay tinatawag na *reference*. Sa halimbawang ito, mayroong dalawang references sa parehong object.

Ang object na may higit sa isang reference ay may higit sa isang pangalan, kaya sinasabi natin na ang object ay *aliased*.

Kung ang aliased object ay mutable, ang mga pagbabagong ginawa gamit ang isang alias ay nakakaapekto sa iba:

```
>>> b[0] = 17
>>> print(a)
[17, 2, 3]
```

Bagaman ang behavior na ito ay maaaring kapaki-pakinabang, ito ay madaling magkamali. Sa pangkalahatan, mas ligtas na iwasan ang aliasing kapag nagtatrabaho ka sa mutable objects.

Para sa immutable objects tulad ng strings, ang aliasing ay hindi gaanong problema. Sa halimbawang ito:

```
a = 'banana'
b = 'banana'
```

halos hindi ito gumagawa ng pagkakaiba kung ang `a` at `b` ay tumutukoy sa parehong string o hindi.

8.13 List arguments

Kapag nagpasa ka ng list sa function, ang function ay nakakakuha ng reference sa list. Kung ang function ay nagmo-modify ng list parameter, nakikita ng caller ang pagbabago. Halimbawa, ang `delete_head` ay nagtatanggal ng unang element mula sa list:

```
def delete_head(t):
    del t[0]
```

Narito kung paano ito ginagamit:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> print(letters)
['b', 'c']
```

Ang parameter na `t` at ang variable na `letters` ay aliases para sa parehong object.

Mahalagang makilala ang pagkakaiba sa pagitan ng operations na nagmo-modify ng lists at operations na gumagawa ng bagong lists. Halimbawa, ang `append` method ay nagmo-modify ng list, pero ang `+` operator ay gumagawa ng bagong list:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print(t1)
[1, 2, 3]
>>> print(t2)
None

>>> t3 = t1 + [3]
>>> print(t3)
[1, 2, 3]
>>> t1 is t3
False
```

Ang pagkakaibang ito ay mahalaga kapag sumusulat ka ng functions na dapat mag-modify ng lists. Halimbawa, ang function na ito *hindi* nagtatanggal ng head ng list:

```
def bad_delete_head(t):
    t = t[1:]               # WRONG!
```

Ang slice operator ay gumagawa ng bagong list at ang assignment ay ginagawang ang `t` ay tumutukoy dito, pero wala sa mga iyon ang may epekto sa list na ipinasa bilang argument.

Ang alternatibo ay sumulat ng function na gumagawa at nagre-return ng bagong list. Halimbawa, ang `tail` ay nagre-return ng lahat maliban sa unang element ng list:

```
def tail(t):
    return t[1:]
```

Ang function na ito ay nag-iiwan sa original list na hindi nabago. Narito kung paano ito ginagamit:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> print(rest)
['b', 'c']
```

Exercise 1: Sumulat ng function na tinatawag na `chop` na tumatanggap ng list at nagmo-modify nito, tinatanggal ang una at huling elements, at nagre-return ng `None`. Pagkatapos sumulat ng function na tinatawag na `middle` na tumatanggap ng list at nagre-return ng bagong list na naglalaman ng lahat maliban sa una at huling elements.

8.14 Debugging

Ang pabayang paggamit ng lists (at iba pang mutable objects) ay maaaring magdulot ng mahabang oras ng debugging. Narito ang ilang karaniwang pitfalls at paraan para iwasan ang mga ito:

1. Huwag kalimutan na karamihan ng list methods ay nagmo-modify ng argument at nagre-return ng `None`. Ito ay kabaligtaran ng string methods, na nagre-return ng bagong string at nag-iiwan sa original.

   Kung sanay ka sa pagsusulat ng string code tulad nito:

   ```
   word = word.strip()
   ```

   Nakakaakit na sumulat ng list code tulad nito:

   ```
   t = t.sort()            # WRONG!
   ```

   Dahil ang `sort` ay nagre-return ng `None`, ang susunod na operation na gagawin mo sa `t` ay malamang na mabibigo.

   Bago gamitin ang list methods at operators, dapat mong basahin nang mabuti ang documentation at pagkatapos i-test ang mga ito sa interactive mode. Ang methods at operators na ibinabahagi ng lists sa iba pang sequences (tulad ng strings) ay na-document sa:

   docs.python.org/library/stdtypes.html#common-sequence-operations

   Ang methods at operators na applicable lang sa mutable sequences ay na-document sa:

   docs.python.org/library/stdtypes.html#mutable-sequence-types

2. Pumili ng idiom at manatili dito.

   Parte ng problema sa lists ay napakaraming paraan para gawin ang mga bagay. Halimbawa, para tanggalin ang element mula sa list, maaari mong gamitin ang `pop`, `remove`, `del`, o kahit slice assignment.

   Para magdagdag ng element, maaari mong gamitin ang `append` method o ang `+` operator. Pero huwag kalimutan na ang mga ito ay tama:

   ```
   t.append(x)
   t = t + [x]
   ```

   At ang mga ito ay mali:

   ```
   t.append([x])          # WRONG!
   t = t.append(x)        # WRONG!
   t + [x]                # WRONG!
   t = t + x              # WRONG!
   ```

   Subukan ang bawat isa sa mga halimbawang ito sa interactive mode para masiguro na naiintindihan mo ang ginagawa nila. Pansinin na ang huli lang ang nagdudulot ng runtime error; ang iba pang tatlo ay legal, pero mali ang ginagawa nila.

3. Gumawa ng copies para iwasan ang aliasing.

   Kung gusto mong gamitin ang method tulad ng `sort` na nagmo-modify ng argument, pero kailangan mo ring panatilihin ang original list, maaari kang gumawa ng kopya.

   ```
   orig = t[:]
   t.sort()
   ```

   Sa halimbawang ito maaari mo ring gamitin ang built-in function na `sorted`, na nagre-return ng bagong, sorted list at nag-iiwan sa original. Pero sa kasong iyon dapat mong iwasan ang paggamit ng `sorted` bilang variable name!

4. Lists, `split`, at files

   Kapag nagbabasa at nagpa-parse tayo ng files, maraming pagkakataon na makatagpo ng input na maaaring i-crash ang program natin kaya magandang ideya na muling bisitahin ang *guardian* pattern pagdating sa pagsusulat ng programs na nagbabasa sa file at naghahanap ng "needle in the haystack".

   Muling bisitahin natin ang program natin na naghahanap ng araw ng linggo sa from lines ng file natin:

   ```
   From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
   ```

   Dahil hinahati natin ang linyang ito sa mga salita, maaari nating alisin ang paggamit ng `startswith` at simpleng tingnan ang unang salita ng linya para matukoy kung interesado tayo sa linya. Maaari nating gamitin ang `continue` para laktawan ang mga linyang walang "From" bilang unang salita tulad ng sumusunod:

```
fhand = open('mbox-short.txt')
for line in fhand:
    words = line.split()
    if words[0] != 'From' : continue
    print(words[2])
```

Mukhang mas simple ito at hindi na natin kailangan gawin ang `rstrip` para tanggalin ang newline sa dulo ng file. Pero mas mabuti ba ito?

```
python search8.py
Sat
Traceback (most recent call last):
  File "search8.py", line 5, in <module>
    if words[0] != 'From' : continue
IndexError: list index out of range
```

Medyo gumagana ito at nakikita natin ang araw mula sa unang linya (Sat), pero pagkatapos nabibigo ang program na may traceback error. Ano ang naging mali? Ano ang messed-up data na nagdulot na ang elegant, clever, at napaka-Pythonic program natin ay mabigo?

Maaari mong titigan ito nang mahabang panahon at pag-isipan o magtanong sa iba para sa tulong, pero ang mas mabilis at mas matalinong approach ay magdagdag ng `print` statement. Ang pinakamabuting lugar para magdagdag ng print statement ay mismo bago ang linya kung saan nabigo ang program at mag-print ng data na tila nagdudulot ng pagkabigo.

Ngayon ang approach na ito ay maaaring gumawa ng maraming linya ng output, pero hindi bababa ay mayroon ka agad na clue tungkol sa problema. Kaya nagdadagdag tayo ng print ng variable na `words` mismo bago ang line five. Nagdadagdag pa tayo ng prefix na "Debug:" sa linya para ma-separate natin ang regular output natin mula sa debug output natin.

```
for line in fhand:
    words = line.split()
    print('Debug:', words)
    if words[0] != 'From' : continue
    print(words[2])
```

Kapag tumatakbo ang program, maraming output ang nag-scroll sa screen pero sa dulo, nakikita natin ang debug output natin at ang traceback kaya alam natin kung ano ang nangyari mismo bago ang traceback.

```
Debug: ['X-DSPAM-Confidence:', '0.8475']
Debug: ['X-DSPAM-Probability:', '0.0000']
Debug: []
Traceback (most recent call last):
  File "search9.py", line 6, in <module>
    if words[0] != 'From' : continue
IndexError: list index out of range
```

Ang bawat debug line ay nagpi-print ng list ng mga salita na nakukuha natin kapag `split` natin ang linya sa mga salita. Kapag nabibigo ang program, ang list ng mga salita ay empty `[]`. Kung bubuksan natin ang file sa text editor at titingnan ang file, sa puntong iyon mukhang ganito:

```
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Sat Jan  5 09:14:16 2008
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

Ang error ay nangyayari kapag ang program natin ay nakatagpo ng blank line! Siyempre mayroong "zero words" sa blank line. Bakit hindi natin naisip iyon kapag sumusulat tayo ng code? Kapag ang code ay naghahanap ng unang salita (`word[0]`) para suriin kung tumutugma ito sa "From", nakakakuha tayo ng "index out of range" error.

Ito siyempre ay perpektong lugar para magdagdag ng ilang *guardian* code para iwasan ang pag-check ng unang salita kung ang unang salita ay wala doon. Mayroong maraming paraan para protektahan ang code na ito; pipiliin nating suriin ang bilang ng mga salita na mayroon tayo bago tingnan ang unang salita:

```
fhand = open('mbox-short.txt')
count = 0
for line in fhand:
    words = line.split()
    # print('Debug:', words)
    if len(words) == 0 : continue
    if words[0] != 'From' : continue
    print(words[2])
```

Una nag-comment out tayo ng debug print statement sa halip na tanggalin ito, kung sakaling mabigo ang modification natin at kailangan nating mag-debug ulit. Pagkatapos nagdagdag tayo ng guardian statement na sumusuri kung mayroon tayong zero words, at kung gayon, ginagamit natin ang `continue` para laktawan ang susunod na linya sa file.

Maaari nating isipin ang dalawang `continue` statements bilang tumutulong sa atin na pinuhin ang set ng mga linya na "interesting" sa atin at na gusto nating i-process pa. Ang linya na walang salita ay "uninteresting" sa atin kaya nilalaktawan natin ang susunod na linya. Ang linya na walang "From" bilang unang salita ay uninteresting sa atin kaya nilalaktawan natin ito.

Ang program na nabago ay tumatakbo nang matagumpay, kaya marahil tama ito. Ang guardian statement natin ay sinisiguro na ang `words[0]` ay hindi kailanman mabibigo, pero marahil hindi ito sapat. Kapag nagpo-program tayo, dapat palaging iniisip natin, "Ano ang maaaring maging mali?"

Exercise 2: Alamin kung aling linya ng program sa itaas ang hindi pa rin properly guarded. Tingnan kung maaari mong gumawa ng text file na nagdudulot na ang program ay mabigo at pagkatapos baguhin ang program para ang linya ay properly guarded at i-test ito para masiguro na ha-handle nito ang bagong text file mo.

Exercise 3: Muling isulat ang guardian code sa halimbawa sa itaas nang walang dalawang `if` statements. Sa halip, gumamit ng compound logical expression gamit ang `or` logical operator na may isang `if` statement.

8.15 Glossary

aliasing Kalagayan kung saan ang dalawa o higit pang variables ay tumutukoy sa parehong object.
delimiter Character o string na ginagamit para ipahiwatig kung saan dapat hatiin ang string.
element Isa sa mga values sa list (o iba pang sequence); tinatawag din na items.
equivalent May parehong value.
index Integer value na tumutukoy sa element sa list.
identical Parehong object (na nagpapahiwatig ng equivalence).
list Sequence ng values.
list traversal Ang sequential na pag-access sa bawat element sa list.
nested list List na element ng ibang list.
object Isang bagay na maaaring tinukoy ng variable. Ang object ay may type at value.
reference Ang asosasyon sa pagitan ng variable at value nito.

8.16 Exercises

Exercise 4: Find all unique words in a file

Gumamit si Shakespeare ng higit sa 20,000 salita sa kanyang mga gawa. Pero paano mo matutukoy iyon? Paano mo gagawin ang list ng lahat ng salitang ginamit ni Shakespeare? I-do-download mo ba lahat ng kanyang gawa, basahin ito at subaybayan ang lahat ng unique words nang manu-mano?

Gamitin natin ang Python para makamit iyon sa halip. I-list ang lahat ng unique words, na naka-sort sa alphabetical order, na naka-store sa file na `romeo.txt` na naglalaman ng subset ng gawa ni Shakespeare.

Para makapagsimula, i-download ang kopya ng file www.py4e.com/code3/romeo.txt. Gumawa ng list ng unique words, na maglalaman ng final result. Sumulat ng program para buksan ang file na `romeo.txt` at basahin ito nang linya sa linya. Para sa bawat linya, hatiin ang linya sa list ng mga salita gamit ang `split` function. Para sa bawat salita, suriin kung ang salita ay nasa list na ng unique words. Kung ang salita ay wala sa list ng unique words, idagdag ito sa list. Kapag natapos ang program, i-sort at i-print ang list ng unique words sa alphabetical order.

```
Enter file: romeo.txt
['Arise', 'But', 'It', 'Juliet', 'Who', 'already',
'and', 'breaks', 'east', 'envious', 'fair', 'grief',
'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft',
'sun', 'the', 'through', 'what', 'window',
'with', 'yonder']
```

Exercise 5: Minimalist Email Client.

Ang MBOX (mail box) ay popular na file format para i-store at ibahagi ang koleksyon ng emails. Ito ay ginamit ng mga naunang email servers at desktop apps. Nang hindi pumapasok sa masyadong maraming detalye, ang MBOX ay text file,

na nag-i-store ng emails nang sunud-sunod. Ang mga emails ay pinaghihiwalay ng espesyal na linya na nagsisimula sa `From` (pansinin ang space). Mahalaga, ang mga linyang nagsisimula sa `From:` (pansinin ang colon) ay naglalarawan sa email mismo at hindi kumikilos bilang separator. Isipin na sumulat ka ng minimalist email app, na nagli-list ng email ng mga sender sa Inbox ng user at nagbi-bilang ng bilang ng emails.

Sumulat ng program para magbasa sa mail box data at kapag nakakita ka ng linya na nagsisimula sa "From", hahatiin mo ang linya sa mga salita gamit ang `split` function. Interesado tayo sa kung sino ang nagpadala ng mensahe, na siyang pangalawang salita sa From line.

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

I-parse mo ang From line at mag-print ng pangalawang salita para sa bawat From line, pagkatapos magbi-bilang ka rin ng bilang ng From (hindi From:) lines at mag-print ng bilang sa dulo. Ito ay magandang sample output na may ilang linya na tinanggal:

```
python fromcount.py
Enter a file name: mbox-short.txt
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu

[...some output removed...]

ray@media.berkeley.edu
cwen@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
There were 27 lines in the file with From as the first word
```

Exercise 6:

Muling isulat ang program na nagpo-prompt sa user para sa list ng numbers at nagpi-print ng maximum at minimum ng numbers sa dulo kapag ang user ay nag-e-enter ng "done". Sumulat ng program para i-store ang numbers na ini-enter ng user sa list at gamitin ang `max()` at `min()` functions para mag-compute ng maximum at minimum numbers pagkatapos matapos ang loop.

```
Enter a number: 6
Enter a number: 2
Enter a number: 9
Enter a number: 3
Enter a number: 5
Enter a number: done
Maximum: 9.0
Minimum: 2.0
```

Chapter 9

Dictionaries

Ang *dictionary* ay katulad ng list, pero mas pangkalahatan. Sa list, ang index positions ay dapat integers; sa dictionary, ang indices ay maaaring (halos) anumang type.

Maaari mong isipin ang dictionary bilang mapping sa pagitan ng set ng indices (na tinatawag na *keys*) at set ng values. Ang bawat key ay nagma-map sa value. Ang asosasyon ng key at value ay tinatawag na *key-value pair* o minsan *item*.

Bilang halimbawa, gagawa tayo ng dictionary na nagma-map mula sa English patungo sa Spanish words, kaya ang keys at values ay lahat strings.

Ang function na `dict` ay gumagawa ng bagong dictionary na walang items. Dahil ang `dict` ay pangalan ng built-in function, dapat mong iwasan ang paggamit nito bilang variable name.

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

Ang curly brackets, `{}`, ay kumakatawan sa empty dictionary. Para magdagdag ng items sa dictionary, maaari mong gamitin ang square brackets:

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

Ang linyang ito ay gumagawa ng item na nagma-map mula sa key na `'one'` patungo sa value na "uno". Kung magpi-print tayo ng dictionary ulit, nakikita natin ang key-value pair na may colon sa pagitan ng key at value:

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

Ang output format na ito ay input format din. Halimbawa, maaari kang gumawa ng bagong dictionary na may tatlong items.

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

Simula sa Python 3.7x ang order ng key-value pairs ay pareho sa input order nila, i.e. ang dictionaries ay ordered structures na ngayon.

Pero hindi talaga mahalaga iyon dahil ang elements ng dictionary ay hindi kailanman na-index gamit ang integer indices. Sa halip, ginagamit mo ang keys para hanapin ang katumbas na values:

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

Ang key na `'two'` ay palaging nagma-map sa value na "dos" kaya ang order ng items ay hindi mahalaga.

Kung ang key ay wala sa dictionary, makakakuha ka ng exception:

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

Ang `len` function ay gumagana sa dictionaries; nagre-return ito ng bilang ng key-value pairs:

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

Ang `in` operator ay gumagana sa dictionaries; sinasabi nito sa iyo kung lumalabas ang isang bagay bilang *key* sa dictionary (ang paglitaw bilang value ay hindi sapat).

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

Para makita kung lumalabas ang isang bagay bilang value sa dictionary, maaari mong gamitin ang method na `values`, na nagre-return ng values bilang type na maaaring i-convert sa list, at pagkatapos gamitin ang `in` operator:

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

Ang `in` operator ay gumagamit ng iba't ibang algorithms para sa lists at dictionaries. Para sa lists, gumagamit ito ng linear search algorithm. Habang ang list ay nagiging mas mahaba, ang search time ay nagiging mas mahaba nang

direkta sa proporsyon sa haba ng list. Para sa dictionaries, ang Python ay gumagamit ng algorithm na tinatawag na *hash table* na may kapansin-pansing property: ang `in` operator ay tumatagal ng halos parehong dami ng panahon anuman ang bilang ng items sa dictionary. Hindi ko ipapaliwanag kung bakit ang hash functions ay napaka-magical, pero maaari kang magbasa pa tungkol dito sa wikipedia.org/wiki/Hash_table.[1]

Exercise 1: I-download ang kopya ng file

www.py4e.com/code3/words.txt

Sumulat ng program na nagbabasa ng mga salita sa *words.txt* at nag-i-store sa kanila bilang keys sa dictionary. Hindi mahalaga kung ano ang values ay. Pagkatapos maaari mong gamitin ang `in` operator bilang mabilis na paraan para suriin kung ang string ay nasa dictionary.

9.1 Dictionary as a set of counters

Ipagpalagay na binigyan ka ng string at gusto mong bilangin kung ilang beses lumalabas ang bawat letra. Mayroong ilang paraan na maaari mong gawin:

1. Maaari kang gumawa ng 26 variables, isa para sa bawat letra ng alphabet. Pagkatapos maaari mong daanan ang string at, para sa bawat character, i-increment ang katumbas na counter, marahil gamit ang chained conditional.

2. Maaari kang gumawa ng list na may 26 elements. Pagkatapos maaari mong i-convert ang bawat character sa number (gamit ang built-in function na `ord`), gamitin ang number bilang index sa list, at i-increment ang naaangkop na counter.

3. Maaari kang gumawa ng dictionary na may characters bilang keys at counters bilang katumbas na values. Sa unang pagkakataon na makikita mo ang character, magdadagdag ka ng item sa dictionary. Pagkatapos i-increment mo ang value ng existing item.

Ang bawat isa sa mga opsyon na ito ay gumagawa ng parehong computation, pero ang bawat isa sa kanila ay nag-i-implement ng computation na iyon sa ibang paraan.

Ang *implementation* ay paraan ng paggawa ng computation; ang ilang implementations ay mas mabuti kaysa sa iba. Halimbawa, ang advantage ng dictionary implementation ay hindi natin kailangang malaman nang maaga kung aling mga letra ang lumalabas sa string at kailangan lang nating gumawa ng lugar para sa mga letrang lumalabas.

Narito kung ano ang hitsura ng code:

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

[1] Kung gusto mong matuto pa tungkol sa hash tables, mayroong course sa https://www.cc4e.com na nag-e-explore kung paano ang programming language na C ay nag-i-implement ng Python dictionary.

```
for c in word:
    if c not in d:
        d[c] = 1
    else:
        d[c] = d[c] + 1
print(d)
```

Epektibong nagco-compute tayo ng *histogram*, na statistical term para sa set ng counters (o frequencies).

Ang `for` loop ay dumadaan sa string. Sa bawat pagkakataon sa loop, kung ang `c` ay wala sa dictionary, gumagawa tayo ng bagong item na may key na `c` at initial value na 1 (dahil nakita na natin ang letrang ito nang isang beses). Kung ang `c` ay nasa dictionary na, i-increment natin ang `d[c]`.

Narito ang output ng program:

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

Ang histogram ay nagpapahiwatig na ang mga letrang "a" at "b" ay lumalabas nang isang beses; ang "o" ay lumalabas nang dalawang beses, at iba pa.

Ang Dictionaries ay may method na tinatawag na `get` na tumatanggap ng key at default value. Kung ang key ay lumalabas sa dictionary, ang `get` ay nagre-return ng katumbas na value; kung hindi, nagre-return ito ng default value. Halimbawa:

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

Maaari nating gamitin ang `get` para sumulat ng histogram loop natin nang mas maikli. Dahil ang `get` method ay awtomatikong nagha-handle ng kaso kung saan ang key ay wala sa dictionary, maaari nating bawasan ang apat na linya sa isa at alisin ang `if` statement.

```
word = 'brontosaurus'
d = dict()
for c in word:
    d[c] = d.get(c,0) + 1
print(d)
```

Ang paggamit ng `get` method para gawing simple ang counting loop na ito ay nagiging napakakaraniwang ginagamit na "idiom" sa Python at gagamitin natin ito nang maraming beses sa natitirang bahagi ng libro. Kaya dapat mong maglaan ng sandali at ihambing ang loop na gumagamit ng `if` statement at `in` operator sa loop na gumagamit ng `get` method. Parehong eksaktong ginagawa nila ang parehong bagay, pero ang isa ay mas maikli.

9.2 Dictionaries and files

Isa sa karaniwang gamit ng dictionary ay bilangin ang occurrence ng mga salita sa file na may nakasulat na teksto. Magsimula tayo sa napakasimpleng file ng mga salita na kinuha mula sa teksto ng *Romeo and Juliet*.

Para sa unang set ng halimbawa, gagamitin natin ang pinaikli at pinasimpleng bersyon ng teksto na walang punctuation. Mamaya magtatrabaho tayo sa teksto ng scene na may kasamang punctuation.

```
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
```

Susulat tayo ng Python program para magbasa sa mga linya ng file, hatiin ang bawat linya sa list ng mga salita, at pagkatapos mag-loop sa bawat isa sa mga salita sa linya at bilangin ang bawat salita gamit ang dictionary.

Makikita mo na mayroon tayong dalawang `for` loops. Ang outer loop ay nagbabasa ng mga linya ng file at ang inner loop ay nag-i-iterate sa bawat isa sa mga salita sa partikular na linya. Ito ay halimbawa ng pattern na tinatawag na *nested loops* dahil ang isa sa mga loops ay ang *outer* loop at ang ibang loop ay ang *inner* loop.

Dahil ang inner loop ay nag-e-execute ng lahat ng iterations nito sa bawat pagkakataon na ang outer loop ay gumagawa ng isang iteration, iniisip natin ang inner loop bilang nag-i-iterate nang "mas mabilis" at ang outer loop bilang nag-i-iterate nang mas mabagal.

Ang kombinasyon ng dalawang nested loops ay sinisiguro na mabibilang natin ang bawat salita sa bawat linya ng input file.

```
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)

# Code: https://www.py4e.com/code3/count1.py
```

Sa `else` statement natin, ginagamit natin ang mas compact na alternatibo para mag-increment ng variable. Ang `counts[word] += 1` ay katumbas ng `counts[word] = counts[word] + 1`. Ang alinmang method ay maaaring gamitin para baguhin ang value ng variable sa anumang nais na dami. Katulad na alternatibo ang umiiral para sa `-=`, `*=`, at `/=`.

Kapag tumatakbo ang program, nakikita natin ang raw dump ng lahat ng counts sa unsorted hash order. (ang file na *romeo.txt* ay available sa www.py4e.com/code3/romeo.txt)

```
python count1.py
Enter the file name: romeo.txt
{'But': 1, 'soft': 1, 'what': 1, 'light': 1, 'through': 1, 'yonder': 1,
'window': 1, 'breaks': 1, 'It': 1, 'is': 3, 'the': 3, 'east': 1, 'and': 3,
'Juliet': 1, 'sun': 2, 'Arise': 1, 'fair': 1, 'kill': 1, 'envious': 1,
'moon': 1, 'Who': 1, 'already': 1, 'sick': 1, 'pale': 1, 'with': 1,
'grief': 1}
```

Medyo hindi maginhawa na tumingin sa dictionary para hanapin ang pinaka karaniwang mga salita at ang kanilang counts, kaya kailangan nating magdagdag ng higit pang Python code para makakuha ng output na mas makakatulong.

9.3 Looping and dictionaries

Kung gagamitin mo ang dictionary bilang sequence sa `for` statement, dumadaan ito sa keys ng dictionary. Ang loop na ito ay nagpi-print ng bawat key at katumbas na value:

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

Narito ang hitsura ng output:

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

Muli, ang keys ay ordered.

Maaari nating gamitin ang pattern na ito para i-implement ang iba't ibang loop idioms na na-describe natin kanina. Halimbawa kung gusto nating hanapin ang lahat ng entries sa dictionary na may value na higit sa sampu, maaari tayong sumulat ng sumusunod na code:

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

Ang `for` loop ay nag-i-iterate sa *keys* ng dictionary, kaya dapat nating gamitin ang index operator para kunin ang katumbas na *value* para sa bawat key. Narito ang hitsura ng output:

```
annie 42
jan 100
```

Nakikita lang natin ang entries na may value na higit sa 10.

Kung gusto mong mag-print ng keys sa alphabetical order, una gumawa ka ng list ng keys sa dictionary gamit ang `keys` method na available sa dictionary objects, at pagkatapos i-sort ang list na iyon at mag-loop sa sorted list, naghahanap ng bawat key at nagpi-print ng key-value pairs sa sorted order tulad ng sumusunod:

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

Here's what the output looks like:

```
['chuck', 'annie', 'jan']
['annie', 'chuck', 'jan']
annie 42
chuck 1
jan 100
```

First you see the list of keys in non-alphabetical order that we get from the `keys` method. Then we see the key-value pairs in alphabetical order from the `for` loop.

9.4 Advanced text parsing

In the above example using the file *romeo.txt*, we made the file as simple as possible by removing all punctuation by hand. The actual text has lots of punctuation, as shown below.

```
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,
```

Since the Python `split` function looks for spaces and treats words as tokens separated by spaces, we would treat the words "soft!" and "soft" as *different* words and create a separate dictionary entry for each word.

Also since the file has capitalization, we would treat "who" and "Who" as different words with different counts.

We can solve both these problems by using the string methods `lower`, `punctuation`, and `translate`. The `translate` is the most subtle of the methods. Here is the documentation for `translate`:

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

Replace the characters in `fromstr` *with the character in the same position in* `tostr` *and delete all characters that are in* `deletestr`*. The* `fromstr` *and* `tostr` *can be empty strings and the* `deletestr` *parameter can be omitted.*

We will not specify the `tostr` but we will use the `deletestr` parameter to delete all of the punctuation. We will even let Python tell us the list of characters that it considers "punctuation":

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

The parameters used by `translate` were different in Python 2.0.

We make the following modifications to our program:

```
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()
    # First two parameters are empty strings
    line = line.translate(line.maketrans("", "", string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)

# Code: https://www.py4e.com/code3/count2.py
```

Part of learning the "Art of Python" or "Thinking Pythonically" is realizing that Python often has built-in capabilities for many common data analysis problems.

Over time, you will see enough example code and read enough of the documentation to know where to look to see if someone has already written something that makes your job much easier.

The following is an abbreviated version of the output:

```
Enter the file name: romeo-full.txt
{'romeo': 40, 'and': 42, 'juliet': 32, 'act': 1, '2': 2, 'scene': 2,
'ii': 1, 'capulets': 1, 'orchard': 2, 'enter': 1, 'he': 5, 'jests': 1,
'at': 9, 'scars': 1, 'that': 30, 'never': 2, 'felt': 1, 'a': 24,
'wound': 1, 'appears': 1, 'above': 6, 'window': 2, 'but': 18,
'soft': 1, 'what': 11, 'light': 5, 'through': 2, 'yonder': 2,
'breaks': 1, ...}
```

Looking through this output is still unwieldy and we can use Python to give us exactly what we are looking for, but to do so, we need to learn about Python *tuples*. We will pick up this example once we learn about tuples.

9.5 Debugging

As you work with bigger datasets it can become unwieldy to debug by printing and checking data by hand. Here are some suggestions for debugging large datasets:

Scale down the input If possible, reduce the size of the dataset. For example if the program reads a text file, start with just the first 10 lines, or with the smallest example you can find. You can either edit the files themselves, or (better) modify the program so it reads only the first `n` lines.

If there is an error, you can reduce `n` to the smallest value that manifests the error, and then increase it gradually as you find and correct errors.

Check summaries and types Instead of printing and checking the entire dataset, consider printing summaries of the data: for example, the number of items in a dictionary or the total of a list of numbers.

A common cause of runtime errors is a value that is not the right type. For debugging this kind of error, it is often enough to print the type of a value.

Write self-checks Sometimes you can write code to check for errors automatically. For example, if you are computing the average of a list of numbers, you could check that the result is not greater than the largest element in the list or less than the smallest. This is called a "sanity check" because it detects results that are "completely illogical".

Another kind of check compares the results of two different computations to see if they are consistent. This is called a "consistency check".

Pretty print the output Formatting debugging output can make it easier to spot an error.

Again, time you spend building scaffolding can reduce the time you spend debugging.

9.6 Glossary

dictionary A mapping from a set of keys to their corresponding values.
hashtable The algorithm used to implement Python dictionaries.
hash function A function used by a hashtable to compute the location for a key.

histogram A set of counters.
implementation A way of performing a computation.
item Another name for a key-value pair.
key An object that appears in a dictionary as the first part of a key-value pair.
key-value pair The representation of the mapping from a key to a value.
lookup A dictionary operation that takes a key and finds the corresponding value.

nested loops When there are one or more loops "inside" of another loop. The inner loop runs to completion each time the outer loop runs once.
value An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word "value".

9.7 Exercises

Exercise 2: Write a program that categorizes each mail message by which day of the week the commit was done. To do this look for lines that start with "From", then look for the third word and keep a running count of each of the days of the week. At the end of the program print out the contents of your dictionary (order does not matter).

Sample Line:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Sample Execution:

```
python dow.py
Enter a file name: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

Exercise 3: Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.

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

Exercise 4: Add code to the above program to figure out who has sent the most messages in the file. After all the data has been read and the dictionary has been created, look through the dictionary using a maximum loop (see Chapter 5: Maximum and minimum loops) to find who has the most messages and print how many messages the person has.

```
Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

Exercise 5: This program records the domain name (instead of the address) where the message was sent from instead of who the mail came from (i.e., the whole email address). At the end of the program, print out the contents of your dictionary.

```
python schoolcount.py
Enter a file name: mbox-short.txt
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```

Chapter 10

Tuples

10.1 Tuples are immutable

Ang tuple[1] ay sequence ng values na katulad ng list. Ang mga values na naka-store sa tuple ay maaaring anumang type, at sila ay na-index ng integers. Ang importanteng pagkakaiba ay ang tuples ay *immutable*. Ang tuples ay *comparable* din at *hashable* kaya maaari nating i-sort ang lists ng mga ito at gamitin ang tuples bilang key values sa Python dictionaries.

Syntactically, ang tuple ay comma-separated list ng values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Bagaman hindi ito kailangan, karaniwang i-enclose ang tuples sa parentheses para matulungan tayong mabilis na makilala ang tuples kapag tinitingnan natin ang Python code:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Para gumawa ng tuple na may isang element, kailangan mong isama ang final comma:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Kung walang comma ang Python ay tinatrato ang `('a')` bilang expression na may string sa parentheses na nag-e-evaluate sa string:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

[1] Fun fact: Ang salitang "tuple" ay nagmula sa mga pangalan na ibinigay sa sequences ng mga numero na may iba't ibang haba: single, double, triple, quadruple, quintuple, sextuple, septuple, etc.

Ang isa pang paraan para gumawa ng tuple ay ang built-in function na `tuple`. Kung walang argument, gumagawa ito ng empty tuple:

```
>>> t = tuple()
>>> print(t)
()
```

Kung ang argument ay sequence (string, list, o tuple), ang result ng pagtawag sa `tuple` ay tuple na may elements ng sequence:

```
>>> t = tuple('lupins')
>>> print(t)
('l', 'u', 'p', 'i', 'n', 's')
```

Dahil ang `tuple` ay pangalan ng constructor, dapat mong iwasan ang paggamit nito bilang variable name.

Karamihan ng list operators ay gumagana din sa tuples. Ang bracket operator ay nag-i-index ng element:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

At ang slice operator ay pumipili ng range ng elements.

```
>>> print(t[1:3])
('b', 'c')
```

Pero kung susubukan mong baguhin ang isa sa elements ng tuple, makakakuha ka ng error:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Hindi mo maaaring baguhin ang elements ng tuple, pero maaari mong palitan ang isang tuple ng iba pa:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

10.2 Comparing tuples

Ang comparison operators ay gumagana sa tuples at iba pang sequences. Ang Python ay nagsisimula sa pag-compare ng unang element mula sa bawat sequence. Kung sila ay equal, nagpapatuloy ito sa susunod na element, at iba pa, hanggang makita ang elements na magkaiba. Ang mga susunod na elements ay hindi isinasaalang-alang (kahit na talagang malaki sila).

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Ang `sort` function ay gumagana sa parehong paraan. Nagso-sort ito primarily sa pamamagitan ng unang element, pero sa kaso ng tie, nagso-sort ito sa pamamagitan ng pangalawang element, at iba pa.

Ang feature na ito ay nagpapahintulot sa pattern na tinatawag na *DSU* para sa

Decorate sequence sa pamamagitan ng paggawa ng list ng tuples na may isa o higit pang sort keys na nauuna sa elements mula sa sequence,
Sort ang list ng tuples gamit ang Python built-in `sort`, at
Undecorate sa pamamagitan ng pagkuha ng sorted elements ng sequence.

Halimbawa, ipagpalagay na mayroon kang list ng mga salita at gusto mong i-sort ang mga ito mula pinakamahaba hanggang pinakamaikli:

```
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list()
for word in words:
    t.append((len(word), word))

t.sort(reverse=True)

res = list()
for length, word in t:
    res.append(word)

print(res)

# Code: https://www.py4e.com/code3/soft.py
```

Ang unang loop ay gumagawa ng list ng tuples, kung saan ang bawat tuple ay salita na nauunahan ng haba nito.

Ang `sort` ay nagko-compare ng unang element, haba, muna, at isinasaalang-alang lang ang pangalawang element para masira ang ties. Ang keyword argument na `reverse=True` ay nagsasabi sa `sort` na pumunta sa decreasing order.

Ang pangalawang loop ay dumadaan sa list ng tuples at gumagawa ng list ng mga salita sa descending order ng haba. Ang apat-na-character na salita ay na-sort sa *reverse* alphabetical order, kaya ang “what” ay lumalabas bago ang “soft” sa sumusunod na list.

Ang output ng program ay ganito:

```
['yonder', 'window', 'breaks', 'light', 'what',
'soft', 'but', 'in']
```

Siyempre ang linya ay nawawalan ng maraming poetic impact kapag naging Python list at na-sort sa descending word length order.

10.3 Tuple assignment

Isa sa mga natatanging syntactic features ng Python language ay ang kakayahang magkaroon ng tuple sa kaliwang bahagi at sequence sa kanang bahagi ng assignment statement. Ito ay nagpapahintulot sa iyo na mag-assign ng higit sa isang variable sa isang pagkakataon sa ibinigay na sequence.

Sa halimbawang ito mayroon tayong two-element tuple at nag-a-assign ng una at pangalawang elements ng tuple sa variables na `x` at `y` sa isang statement.

```
>>> m = ( 'have', 'fun' )
>>> x, y = m
>>> x
'have'
>>> y
'fun'
>>>
```

Ito ay mas pangkalahatan kaysa sa tuple-to-tuple assignment. Parehong tuples at lists ay sequences, kaya ang syntax na ito ay gumagana din sa two element list.

```
>>> m = [ 'have', 'fun' ]
>>> x, y = m
>>> x
'have'
>>> y
'fun'
>>>
```

Hindi ito magic, ang Python ay *roughly* nagta-translate ng tuple assignment syntax na maging sumusunod:[2]

```
>>> m = ( 'have', 'fun' )
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
>>>
```

Stylistically kapag gumagamit tayo ng tuple sa kaliwang bahagi ng assignment statement, tinatanggal natin ang parentheses, pero ang sumusunod ay pantay na valid syntax:

```
>>> m = ( 'have', 'fun' )
>>> (x, y) = m
```

[2]Ang Python ay hindi literal na nagta-translate ng syntax. Halimbawa, kung susubukan mo ito sa dictionary, hindi ito gagana tulad ng inaasahan mo.

```
>>> x
'have'
>>> y
'fun'
>>>
```

Ang partikular na matalinong application ng tuple assignment ay nagpapahintulot sa atin na *swap* ang values ng dalawang variables sa isang statement:

```
>>> a, b = b, a
```

Ang parehong bahagi ng statement na ito ay tuples, pero ang kaliwang bahagi ay tuple ng variables; ang kanang bahagi ay tuple ng expressions. Ang bawat value sa kanang bahagi ay na-a-assign sa kani-kaniyang variable sa kaliwang bahagi. Lahat ng expressions sa kanang bahagi ay na-e-evaluate bago ang alinman sa assignments.

Ang bilang ng variables sa kaliwa at ang bilang ng values sa kanan ay dapat pareho:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Mas pangkalahatan, ang kanang bahagi ay maaaring anumang uri ng sequence (string, list, o tuple). Halimbawa, para hatiin ang email address sa user name at domain, maaari mong isulat:

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

Ang return value mula sa `split` ay list na may dalawang elements; ang unang element ay na-a-assign sa `uname`, ang pangalawa sa `domain`.

```
>>> print(uname)
monty
>>> print(domain)
python.org
```

10.4 Dictionaries and tuples

Ang dictionaries ay may method na tinatawag na `items` na nagre-return ng list ng tuples, kung saan ang bawat tuple ay key-value pair:

```
>>> d = {'b':1, 'a':10, 'c':22}
>>> t = list(d.items())
>>> print(t)
[('b', 1), ('a', 10), ('c', 22)]
```

Tulad ng dapat mong asahan mula sa dictionary, ang items ay nasa non-alphabetical order.

Gayunpaman, dahil ang list ng tuples ay list, at ang tuples ay comparable, maaari na nating i-sort ang list ng tuples. Ang pagko-convert ng dictionary sa list ng tuples ay paraan para sa atin na i-output ang contents ng dictionary na na-sort ayon sa key:

```
>>> d = {'b':1, 'a':10, 'c':22}
>>> t = list(d.items())
>>> t
[('b', 1), ('a', 10), ('c', 22)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

Ang bagong list ay na-sort sa ascending alphabetical order ayon sa key value.

10.5 Multiple assignment with dictionaries

Sa pagsasama ng `items`, tuple assignment, at `for`, makikita mo ang magandang code pattern para dumaan sa keys at values ng dictionary sa isang loop:

```
d = {'a':10, 'b':1, 'c':22}
for key, val in d.items():
    print(val, key)
```

Ang loop na ito ay may dalawang *iteration variables* dahil ang `items` ay nagre-return ng list ng tuples at ang `key, val` ay tuple assignment na sunud-sunod na nag-i-iterate sa bawat isa sa key-value pairs sa dictionary.

Para sa bawat iteration sa loop, parehong `key` at `val` ay umaabante sa susunod na key-value pair sa dictionary (nasa hash order pa rin).

Ang output ng loop na ito ay:

```
10 a
1 b
22 c
```

Muli, nasa hash key order ito (i.e., walang partikular na order).

Kung pagsasamahin natin ang dalawang techniques na ito, maaari nating i-print ang contents ng dictionary na na-sort ayon sa *value* na naka-store sa bawat key-value pair.

Para gawin ito, una tayong gumagawa ng list ng tuples kung saan ang bawat tuple ay `(value, key)`. Ang `items` method ay magbibigay sa atin ng list ng `(key, value)` tuples, pero sa pagkakataong ito gusto nating mag-sort ayon sa value, hindi key. Kapag nagawa na natin ang list na may value-key tuples, simpleng bagay lang na i-sort ang list sa reverse order at i-print ang bagong, sorted list.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> l = list()
>>> for key, val in d.items() :
...     l.append( (val, key) )
...
>>> l
[(10, 'a'), (1, 'b'), (22, 'c')]
>>> l.sort(reverse=True)
>>> l
[(22, 'c'), (10, 'a'), (1, 'b')]
>>>
```

Sa maingat na paggawa ng list ng tuples para magkaroon ng value bilang unang element ng bawat tuple, maaari nating i-sort ang list ng tuples at makuha ang dictionary contents natin na na-sort ayon sa value.

10.6 The most common words

Bumabalik sa running example natin ng text mula sa *Romeo and Juliet* Act 2, Scene 2, maaari nating dagdagan ang program natin para gamitin ang technique na ito para i-print ang sampung pinakakaraniwang salita sa text tulad ng sumusunod:

```
import string
fhand = open('romeo-full.txt')
counts = dict()
for line in fhand:
    line = line.translate(str.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Sort the dictionary by value
lst = list()
for key, val in list(counts.items()):
    lst.append((val, key))

lst.sort(reverse=True)

for key, val in lst[:10]:
    print(key, val)

# Code: https://www.py4e.com/code3/count3.py
```

Ang unang parte ng program na nagbabasa ng file at nagko-compute ng dictionary na nagma-map ng bawat salita sa bilang ng salita sa dokumento ay hindi nagbago.

Pero sa halip na simpleng mag-print ng `counts` at tapusin ang program, gumagawa tayo ng list ng `(val, key)` tuples at pagkatapos i-sort ang list sa reverse order.

Dahil ang value ay una, ito ay gagamitin para sa comparisons. Kung mayroong higit sa isang tuple na may parehong value, titingnan nito ang pangalawang element (ang key), kaya ang tuples kung saan ang value ay pareho ay mas na-sort ayon sa alphabetical order ng key.

Sa dulo sumusulat tayo ng magandang `for` loop na gumagawa ng multiple assignment iteration at nagpi-print ng sampung pinakakaraniwang salita sa pamamagitan ng pag-i-iterate sa slice ng list (`lst[:10]`).

Kaya ngayon ang output ay mukhang gusto natin para sa word frequency analysis natin.

```
61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee
```

Ang katotohanan na ang complex data parsing at analysis na ito ay maaaring gawin gamit ang madaling maintindihang 19-line Python program ay isa sa mga dahilan kung bakit ang Python ay magandang pagpipilian bilang language para mag-explore ng information.

10.7 Using tuples as keys in dictionaries

Dahil ang tuples ay *hashable* at ang lists ay hindi, kung gusto nating gumawa ng *composite* key para gamitin sa dictionary kailangan nating gumamit ng tuple bilang key.

Makakatagpo tayo ng composite key kung gusto nating gumawa ng telephone directory na nagma-map mula sa last-name, first-name pairs patungo sa telephone numbers. Ipagpalagay na tinukoy na natin ang variables na `last`, `first`, at `number`, maaari tayong sumulat ng dictionary assignment statement tulad ng sumusunod:

```
directory[last,first] = number
```

Ang expression sa brackets ay tuple. Maaari nating gamitin ang tuple assignment sa `for` loop para dumaan sa dictionary na ito.

```
for last, first in directory:
    print(first, last, directory[last,first])
```

Ang loop na ito ay dumadaan sa keys sa `directory`, na tuples. Nag-a-assign ito ng elements ng bawat tuple sa `last` at `first`, pagkatapos nagpi-print ng pangalan at katumbas na telephone number.

10.8 Sequences: strings, lists, and tuples - Oh My!

Nakatuon ako sa lists ng tuples, pero halos lahat ng halimbawa sa chapter na ito ay gumagana din sa lists ng lists, tuples ng tuples, at tuples ng lists. Para maiwasan ang pag-enumerate ng posibleng combinations, minsan mas madaling pag-usapan ang sequences ng sequences.

Sa maraming konteksto, ang iba't ibang uri ng sequences (strings, lists, at tuples) ay maaaring gamitin nang magkakapalit. Kaya paano at bakit pipiliin mo ang isa kaysa sa iba?

Para magsimula sa halata, ang strings ay mas limitado kaysa sa iba pang sequences dahil ang elements ay dapat characters. Immutable din sila. Kung kailangan mo ng kakayahang baguhin ang characters sa string (sa halip na gumawa ng bagong string), maaaring gusto mong gumamit ng list ng characters sa halip.

Ang lists ay mas karaniwan kaysa sa tuples, karamihan dahil mutable sila. Pero may ilang kaso kung saan maaaring mas gusto mo ang tuples:

1. Sa ilang konteksto, tulad ng `return` statement, ito ay syntactically mas simple na gumawa ng tuple kaysa sa list. Sa iba pang konteksto, maaaring mas gusto mo ang list.
2. Kung gusto mong gamitin ang sequence bilang dictionary key, kailangan mong gamitin ang immutable type tulad ng tuple o string.
3. Kung nagpapasa ka ng sequence bilang argument sa function, ang paggamit ng tuples ay binabawasan ang posibilidad ng hindi inaasahang behavior dahil sa aliasing.

Dahil ang tuples ay immutable, hindi sila nagbibigay ng methods tulad ng `sort` at `reverse`, na nagmo-modify ng existing lists. Gayunpaman ang Python ay nagbibigay ng built-in functions na `sorted` at `reversed`, na tumatanggap ng anumang sequence bilang parameter at nagre-return ng bagong sequence na may parehong elements sa ibang order.

10.9 List comprehension

Minsan gusto mong gumawa ng sequence sa pamamagitan ng paggamit ng data mula sa iba pang sequence. Maaari mong makamit ito sa pamamagitan ng pagsusulat ng for loop at pag-a-append ng isang item sa isang pagkakataon. Halimbawa, kung gusto mong i-convert ang list ng strings – bawat string na nag-i-store ng digits – sa mga numero na maaari mong i-sum, susulat ka ng:

```
list_of_ints_in_strings = ['42', '65', '12']
list_of_ints = []
for x in list_of_ints_in_strings:
    list_of_ints.append(int(x))

print(sum(list_of_ints))
```

Gamit ang list comprehension, ang code sa itaas ay maaaring isulat sa mas compact na paraan:

```
list_of_ints_in_strings = ['42', '65', '12']
list_of_ints = [ int(x) for x in list_of_ints_in_strings ]
print(sum(list_of_ints))
```

10.10 Debugging

Ang lists, dictionaries at tuples ay kilala sa pangkalahatan bilang *data structures*; sa chapter na ito nagsisimula tayong makakita ng compound data structures, tulad ng lists ng tuples, at dictionaries na naglalaman ng tuples bilang keys at lists bilang values. Ang compound data structures ay kapaki-pakinabang, pero madali silang magkaroon ng tinatawag kong *shape errors*; iyon ay, errors na sanhi kapag ang data structure ay may maling type, size, o composition, o maaaring sumusulat ka ng code at nakakalimutan ang shape ng iyong data at nagdudulot ng error. Halimbawa, kung umaasahan ka ng list na may isang integer at binigyan kita ng plain old integer (hindi sa list), hindi ito gagana.

10.11 Glossary

comparable Type kung saan ang isang value ay maaaring suriin para makita kung mas malaki, mas maliit, o katumbas ng iba pang value ng parehong type. Ang mga types na comparable ay maaaring ilagay sa list at i-sort.

data structure Koleksyon ng related values, kadalasang inoorganisa sa lists, dictionaries, tuples, etc.

DSU Abbreviation ng "decorate-sort-undecorate", pattern na nagsasangkot ng paggawa ng list ng tuples, pagso-sort, at pagkuha ng parte ng result.

gather Ang operation ng pag-assemble ng variable-length argument tuple.

hashable Type na may hash function. Ang immutable types tulad ng integers, floats, at strings ay hashable; ang mutable types tulad ng lists at dictionaries ay hindi.

scatter Ang operation ng pagtrato sa sequence bilang list ng arguments.

shape (of a data structure) Buod ng type, size, at composition ng data structure.

singleton List (o iba pang sequence) na may isang element.

tuple Immutable sequence ng elements.

tuple assignment Assignment na may sequence sa kanang bahagi at tuple ng variables sa kaliwang bahagi. Ang kanang bahagi ay na-e-evaluate at pagkatapos ang elements nito ay na-a-assign sa variables sa kaliwa.

10.12 Exercises

Exercise 1: Rebisahin ang naunang program tulad ng sumusunod: Basahin at i-parse ang "From" lines at kunin ang addresses mula sa linya. Bilangin ang bilang ng messages mula sa bawat tao gamit ang dictionary.

Pagkatapos mabasa ang lahat ng data, mag-print ng tao na may pinakamaraming commits sa pamamagitan ng paggawa ng list ng (count, email) tuples mula sa dictionary. Pagkatapos i-sort ang list sa reverse order at mag-print ng tao na may pinakamaraming commits.

```
Sample Line:
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

Exercise 2: Ang program na ito ay nagbi-bilang ng distribution ng oras ng araw para sa bawat isa sa messages. Maaari mong kunin ang oras mula sa "From" line sa pamamagitan ng paghahanap ng time string at pagkatapos paghahati ng string na iyon sa mga parte gamit ang colon character. Kapag na-accumulate mo na ang counts para sa bawat oras, mag-print ng counts, isa bawat linya, na na-sort ayon sa oras tulad ng ipinakita sa ibaba.

```
python timeofday.py
Enter a file name: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

Exercise 3: Sumulat ng program na nagbabasa ng file at nagpi-print ng *mga letra* sa decreasing order ng frequency.

Ang iyong program ay dapat i-convert ang lahat ng input sa lower case at bilangin lang ang mga letra a-z. Ang iyong program ay hindi dapat magbilang ng spaces, digits, punctuation, o anumang bagay maliban sa mga letra a-z. Maghanap ng text samples mula sa ilang iba't ibang languages at tingnan kung paano nag-iiba ang letter frequency sa pagitan ng mga languages. I-compare ang iyong results sa tables sa https://wikipedia.org/wiki/Letter_frequencies.

Chapter 11

Regular expressions

Hanggang ngayon, nagbabasa tayo sa mga files, naghahanap ng patterns at kumukuha ng iba't ibang bits ng lines na nakakainteresante sa atin. Gumagamit tayo ng string methods tulad ng `split` at `find` at gumagamit ng lists at string slicing para kunin ang mga parte ng lines.

Ang gawaing ito ng paghahanap at pagkuha ay napakakaraniwan na ang Python ay may napakamakapangyarihang module na tinatawag na *regular expressions* na naghahandle ng marami sa mga gawaing ito nang elegante. Ang dahilan kung bakit hindi namin ipinakilala ang regular expressions mas maaga sa libro ay dahil habang sila ay napakamakapangyarihan, medyo kumplikado sila at ang syntax nila ay nangangailangan ng ilang paggamit para masanay.

Ang regular expressions ay halos sarili nilang maliit na programming language para sa paghahanap at pag-parse ng strings. Sa katunayan, buong mga libro ang nasusulat tungkol sa paksa ng regular expressions. Sa chapter na ito, tatalakayin lang natin ang basics ng regular expressions. Para sa mas detalyado tungkol sa regular expressions, tingnan:

https://en.wikipedia.org/wiki/Regular_expression

https://docs.python.org/library/re.html

Maaari mo ring i-bookmark ang sumusunod na page para mag-eksperimento sa regular expressions habang natututo ka:

https://regex101.com/

Ang regular expression module na `re` ay dapat i-import sa program mo bago mo ito magamit. Ang pinakasimpleng paggamit ng regular expression module ay ang `search()` function. Ang sumusunod na program ay nagpapakita ng trivial na paggamit ng search function.

```
# Search for lines that contain 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line):
```

```
        print(line)

# Code: https://www.py4e.com/code3/re01.py
```

Binubuksan natin ang file, naglo-loop sa bawat linya, at gumagamit ng regular expression na `search()` para lang mag-print ng lines na naglalaman ng string "From:". Ang program na ito ay hindi gumagamit ng tunay na kapangyarihan ng regular expressions, dahil maaari lang nating gamitin ang `line.find()` para makamit ang parehong result.

Ang kapangyarihan ng regular expressions ay dumarating kapag nagdadagdag tayo ng espesyal na characters sa search string na nagpapahintulot sa atin na mas tumpak na kontrolin kung aling lines ang tumugma sa string. Ang pagdaragdag ng espesyal na characters na ito sa aming regular expression ay nagpapahintulot sa atin na gumawa ng sophisticated matching at extraction habang sumusulat ng napakakaunting code.

Halimbawa, ang caret character ay ginagamit sa regular expressions para tumugma sa "simula" ng linya. Maaari nating baguhin ang program natin para lang tumugma sa lines kung saan ang "From:" ay nasa simula ng linya tulad ng sumusunod:

```
# Search for lines that start with 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print(line)

# Code: https://www.py4e.com/code3/re02.py
```

Ngayon ay tumutugma lang tayo sa lines na *nagsisimula sa* string "From:". Ito ay napakasimpleng halimbawa pa rin na maaari nating gawin nang katumbas gamit ang `startswith()` method mula sa string module. Pero nagsisilbi ito para ipakilala ang konsepto na ang regular expressions ay naglalaman ng espesyal na action characters na nagbibigay sa atin ng mas maraming kontrol kung ano ang tutugma sa regular expression.

11.1 Character matching in regular expressions

Mayroong ilang iba pang espesyal na characters na nagpapahintulot sa atin na gumawa ng mas makapangyarihang regular expressions. Ang pinakakaraniwang ginagamit na espesyal character ay ang period o full stop, na tumutugma sa anumang character.

Sa sumusunod na halimbawa, ang regular expression na `F..m:` ay tutugma sa anumang sa mga strings na "From:", "Fxxm:", "F12m:", o "F!@m:" dahil ang period characters sa regular expression ay tumutugma sa anumang character.

```
# Search for lines that start with 'F', followed by
# 2 characters, followed by 'm:'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line):
        print(line)

# Code: https://www.py4e.com/code3/re03.py
```

Ito ay partikular na makapangyarihan kapag pinagsama sa kakayahang ipahiwatig na ang character ay maaaring ulitin ng anumang bilang ng beses gamit ang `*` o `+` characters sa regular expression mo. Ang espesyal na characters na ito ay nangangahulugang sa halip na tumugma sa isang character sa search string, tumutugma sila sa zero-or-more characters (sa kaso ng asterisk) o one-or-more ng characters (sa kaso ng plus sign).

Maaari pa nating paliitin ang lines na tinutugma natin gamit ang paulit-ulit na *wild card* character sa sumusunod na halimbawa:

```
# Search for lines that start with From and have an at sign
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:.+@', line):
        print(line)

# Code: https://www.py4e.com/code3/re04.py
```

Ang search string na `^From:.+@` ay matagumpay na tutugma sa lines na nagsisimula sa "From:", na sinusundan ng isa o higit pang characters (`.+`), na sinusundan ng at-sign. Kaya tutugma ito sa sumusunod na linya:

```
From: stephen.marquard@uct.ac.za
```

Maaari mong isipin ang `.+` wildcard bilang pagpapalawak para tumugma sa lahat ng characters sa pagitan ng colon character at at-sign.

```
From:.+@
```

Magandang isipin ang plus at asterisk characters bilang "pushy" o "greedy." Halimbawa, ang sumusunod na string ay tutugma sa huling at-sign sa string habang ang `.+` ay nagpu-push palabas, ang nasa `cwen@iupui.edu`.

```
From: stephen.marquard@uct.ac.za, csev@umich.edu, and cwen@iupui.edu
```

Posibleng sabihin sa asterisk o plus sign na huwag maging masyadong "greedy." Kung mag-a-append ka ng `?`, para gawin itong `.*?` o `.+?`, ang regex mo ay tutugma sa pinakaunang instance imbes na huling instance. Sa halimbawa sa itaas, ang paggamit ng `.+?@` ay tutugma sa unang at-sign, ang nasa `stephen.marquard@uct.ac.za`. Para sa mas maraming detalye, tingnan ang Python documentation on greedy vs non-greedy quantifiers.

11.2 Extracting data using regular expressions

Kung gusto nating kunin ang data mula sa string sa Python maaari nating gamitin ang `findall()` method para kunin ang lahat ng substrings na tumutugma sa regular expression. Gamitin natin ang halimbawa ng pagkuha ng anumang bagay na mukhang email address mula sa anumang linya anuman ang format. Halimbawa, gusto nating kunin ang email addresses mula sa bawat isa sa sumusunod na lines:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
          for <source@collab.sakaiproject.org>;
Received: (from apache@localhost)
Author: stephen.marquard@uct.ac.za
```

Hindi natin gusto na sumulat ng code para sa bawat uri ng lines, na nagha-hati at nag-slice nang iba para sa bawat linya. Ang sumusunod na program ay gumagamit ng `findall()` para hanapin ang lines na may email addresses sa kanila at kunin ang isa o higit pang addresses mula sa bawat isa sa mga lines na iyon.

```
import re
s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting @2PM'
lst = re.findall(r'\S+@\S+', s)
print(lst)

# Code: https://www.py4e.com/code3/re05.py
```

Ang `findall()` method ay naghahanap sa string sa pangalawang argument at nagre-return ng list ng lahat ng strings na mukhang email addresses. Gumagamit tayo ng two-character sequence na tumutugma sa non-whitespace character (`\S`). Mapapansin mo na nag-append din tayo ng letrang `r` bago lang ang regular expression natin; sinasabi nito sa Python na bigyang-kahulugan ang regular expression natin bilang raw string, at huwag tratuhin ang backslashes bilang escape characters (tulad ng ginagamit natin sa `\n`).

Ang output ng program ay magiging:

```
['csev@umich.edu', 'cwen@iupui.edu']
```

Sa pag-translate ng regular expression, naghahanap tayo ng substrings na may hindi bababa sa isang non-whitespace character, na sinusundan ng at-sign, na sinusundan ng hindi bababa sa isa pang non-whitespace character. Ang `\S+` ay tumutugma sa kasing dami ng non-whitespace characters na posible.

Ang regular expression ay tutugma nang dalawang beses (csev@umich.edu at cwen@iupui.edu), pero hindi ito tutugma sa string na "@2PM" dahil walang non-blank characters *bago* ang at-sign. Maaari nating gamitin ang regular expression na ito sa program para basahin ang lahat ng lines sa file at mag-print ng anumang bagay na mukhang email address tulad ng sumusunod:

```
# Search for lines that have an at sign between characters
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall(r'\S+@\S+', line)
    if len(x) > 0:
        print(x)

# Code: https://www.py4e.com/code3/re06.py
```

Binabasa natin ang bawat linya at pagkatapos kinukuha ang lahat ng substrings na tumutugma sa aming regular expression. Dahil ang `findall()` ay nagre-return ng list, simpleng sinusuri natin kung ang bilang ng elements sa returned list natin ay higit sa zero para mag-print lang ng lines kung saan nakakita tayo ng hindi bababa sa isang substring na mukhang email address.

Kung patakbuhin natin ang program sa *mbox-short.txt* makukuha natin ang sumusunod na output:

```
...
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['cwen@iupui.edu']
['source@collab.sakaiproject.org']
['cwen@iupui.edu']
['cwen@iupui.edu']
['wagnermr@iupui.edu']
```

Ang ilan sa aming email addresses ay may maling characters tulad ng "<" o ";" sa simula o dulo. Sabihin natin na interesado lang tayo sa parte ng string na nagsisimula at nagtatapos sa letra o numero.

Para gawin ito, gumagamit tayo ng iba pang feature ng regular expressions. Ang square brackets ay ginagamit para ipahiwatig ang set ng maraming acceptable characters na handa nating isaalang-alang na tumugma. Sa isang diwa, ang `\S` ay naghihingi na tumugma sa set ng "non-whitespace characters". Ngayon ay magiging mas eksplisito tayo sa mga characters na tutugma natin.

Narito ang bagong regular expression natin:

```
[a-zA-Z0-9]\S*@\S*[a-zA-Z]
```

Ito ay nagiging medyo kumplikado at maaari mong simulan na makita kung bakit ang regular expressions ay sarili nilang maliit na language. Sa pag-translate ng regular expression na ito, naghahanap tayo ng substrings na nagsisimula sa *isang* lowercase letter, uppercase letter, o numero "[a-zA-Z0-9]", na sinusundan ng zero o higit pang non-blank characters (`\S*`), na sinusundan ng at-sign, na sinusundan ng zero o higit pang non-blank characters (`\S*`), na sinusundan ng uppercase o lowercase letter. Tandaan na lumipat tayo mula sa `+` patungo sa `*` para ipahiwatig ang zero o higit pang non-blank characters dahil ang `[a-zA-Z0-9]` ay isa nang non-blank character. Tandaan na ang `*` o `+` ay nalalapat sa isang character na agad nasa kaliwa ng plus o asterisk.

Kung gagamitin natin ang expression na ito sa program natin, mas malinis ang data natin:

```
# Search for lines that have an at sign between characters
# The characters must be a letter or number
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall(r'[a-zA-Z0-9]\S*@\S*[a-zA-Z]', line)
    if len(x) > 0:
        print(x)

# Code: https://www.py4e.com/code3/re07.py
```

```
...
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801032122.m03LMFo4005148@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
```

Mapansin na sa `source@collab.sakaiproject.org` lines, ang regular expression natin ay nagtanggal ng dalawang letra sa dulo ng string (">;"). Ito ay dahil kapag nag-a-append tayo ng `[a-zA-Z]` sa dulo ng regular expression natin, hinihingi natin na anumang string na makikita ng regular expression parser ay dapat magtapos sa letra. Kaya kapag nakikita nito ang ">" sa dulo ng "sakaiproject.org>;" simpleng humihinto ito sa huling "matching" na letra na nakita nito (i.e., ang "g" ay ang huling magandang match).

Tandaan din na ang output ng program ay Python list na may string bilang isang element sa list.

11.3 Combining searching and extracting

Kung gusto nating hanapin ang mga numero sa lines na nagsisimula sa string "X-" tulad ng:

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

hindi lang natin gusto ang anumang floating-point numbers mula sa anumang lines. Gusto lang nating kunin ang mga numero mula sa lines na may syntax sa itaas.

Maaari nating gawin ang sumusunod na regular expression para pumili ng lines:

```
^X-.*: [0-9.]+
```

Sa pag-translate nito, sinasabi natin, gusto natin ng lines na nagsisimula sa `X-`, na sinusundan ng zero o higit pang characters (`.*`), na sinusundan ng colon (`:`) at pagkatapos ng space. Pagkatapos ng space naghahanap tayo ng isa o higit pang characters na alinman sa digit (0-9) o period `[0-9.]+`. Tandaan na sa loob ng square brackets, ang period ay tumutugma sa aktwal na period (i.e., hindi ito wildcard sa pagitan ng square brackets).

Ito ay napakapinid na expression na halos tutugma lang sa lines na interesado tayo tulad ng sumusunod:

```
# Search for lines that start with 'X' followed by any non
# whitespace characters and ':'
# followed by a space and any number.
# The number can include a decimal.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search(r'^X-.*: [0-9.]+', line):
        print(line)

# Code: https://www.py4e.com/code3/re10.py
```

Kapag pinatakbo natin ang program, nakikita natin ang data na maganda ang filter para ipakita lang ang lines na hinahanap natin.

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
...
```

Pero ngayon kailangan nating solusyonan ang problema ng pagkuha ng mga numero. Habang sapat na simple ang paggamit ng `split`, maaari tayong gumamit ng iba pang feature ng regular expressions para parehong maghanap at mag-parse ng linya sa parehong pagkakataon.

Ang parentheses ay isa pang espesyal na character sa regular expressions. Kapag nagdaragdag ka ng parentheses sa regular expression, hindi sila pinapansin kapag tumutugma sa string. Pero kapag gumagamit ka ng `findall()`, ang parentheses ay nagpapahiwatig na habang gusto mo na tumugma ang buong expression, interesado ka lang na kunin ang parte ng substring na tumutugma sa regular expression.

Kaya ginagawa natin ang sumusunod na pagbabago sa program natin:

```
# Search for lines that start with 'X' followed by any
# non whitespace characters and ':' followed by a space
# and any number. The number can include a decimal.
# Then print the number if it is greater than zero.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall(r'^X-.*: ([0-9.]+)', line)
    if len(x) > 0:
        print(x)

# Code: https://www.py4e.com/code3/re11.py
```

Sa halip na tawagin ang `search()`, nagdaragdag tayo ng parentheses sa paligid ng parte ng regular expression na kumakatawan sa floating-point number para ipahiwatig na gusto lang natin na ibigay sa atin ng `findall()` ang floating-point number na parte ng matching string.

Ang output mula sa program na ito ay ganito:

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
...
```

Ang mga numero ay nasa list pa rin at kailangang i-convert mula sa strings patungo sa floating point, pero ginamit na natin ang kapangyarihan ng regular expressions para parehong maghanap at kunin ang impormasyon na nakakainteresante sa atin.

Bilang isa pang halimbawa ng technique na ito, kung titingnan mo ang file mayroong ilang lines ng form:

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

Kung gusto nating kunin ang lahat ng revision numbers (ang integer number sa dulo ng mga lines na ito) gamit ang parehong technique sa itaas, maaari tayong sumulat ng sumusunod na program:

```
# Search for lines that start with 'Details: rev='
# followed by numbers
# Then print the number if one is found
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall(r'^Details:.*rev=([0-9]+)$', line)
    if len(x) > 0:
```

```
        print(x)

# Code: https://www.py4e.com/code3/re12.py
```

Sa pag-translate ng regular expression natin, naghahanap tayo ng lines na nagsisimula sa `Details:`, na sinusundan ng anumang bilang ng characters (`.*`), na sinusundan ng `rev=`, at pagkatapos ng isa o higit pang digits. Gusto nating hanapin ang lines na tumutugma sa buong expression pero gusto lang nating kunin ang integer number sa dulo ng linya, kaya pinalilibutan natin ang `[0-9]+` ng parentheses, at nagdaragdag tayo ng `$` para ipahiwatig na tumutugma partikular sa dulo ng linya.

Kapag pinatakbo natin ang program, makukuha natin ang sumusunod na output:

```
['39772']
['39771']
['39770']
['39769']
...
```

Tandaan na ang `[0-9]+` ay "greedy" at sinusubukan nitong gawing mas malaki ang string ng digits hangga't maaari bago kunin ang mga digits na iyon. Ang "greedy" behavior na ito ang dahilan kung bakit nakukuha natin ang lahat ng limang digits para sa bawat numero. Ang regular expression module ay lumalawak sa parehong direksyon hanggang makakita ng non-digit, o simula o dulo ng linya.

Ngayon maaari nating gamitin ang regular expressions para gawin ulit ang exercise mula sa mas maaga sa libro kung saan interesado tayo sa oras ng araw ng bawat mail message. Naghanap tayo ng lines ng form:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

at gusto nating kunin ang oras ng araw para sa bawat linya. Dati ginawa natin ito gamit ang dalawang tawag sa `split`. Una hinati ang linya sa mga salita at pagkatapos kinuha natin ang ikalimang salita at hinati ulit sa colon character para kunin ang dalawang characters na interesado tayo.

Habang gumagana ito, nagreresulta ito sa medyo brittle code na nag-a-assume na ang lines ay maganda ang format. Kung magdaragdag ka ng sapat na error checking (o malaking try/except block) para siguraduhin na ang program mo ay hindi kailanman mabibigo kapag binigyan ng maling format na lines, ang code ay lalaki sa 10-15 lines ng code na medyo mahirap basahin.

Maaari nating gawin ito sa mas simpleng paraan gamit ang sumusunod na regular expression:

```
^From .* [0-9][0-9]:
```

Ang translation ng regular expression na ito ay naghahanap tayo ng lines na nagsisimula sa `From` (tandaan ang space), na sinusundan ng anumang bilang ng characters (`.*`), na sinusundan ng space, na sinusundan ng dalawang digits `[0-9][0-9]`,

na sinusundan ng colon character. Ito ang definition ng mga uri ng lines na hinahanap natin.

Para kunin lang ang oras gamit ang `findall()`, nagdaragdag tayo ng parentheses sa paligid ng dalawang digits tulad ng sumusunod:

```
^From .* ([0-9][0-9]):
```

Ito ay nagreresulta sa sumusunod na program:

```
# Search for lines that start with From and a character
# followed by a two digit number between 00 and 99 followed by ':'
# Then print the number if one is found
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From .* ([0-9][0-9]):', line)
    if len(x) > 0: print(x)

# Code: https://www.py4e.com/code3/re13.py
```

Kapag tumatakbo ang program, gumagawa ito ng sumusunod na output:

```
['09']
['18']
['16']
['15']
...
```

11.4 Escape character

Ang regular expressions ay gumagamit ng espesyal na characters tulad ng ^ para tumugma sa simula ng linya, $ para sa dulo ng linya, at . bilang wildcard; gayunpaman, minsan gusto nating tumugma sa mga characters na iyon nang literal. Kailangan natin ng paraan para ipahiwatig na gusto nating tumugma sa aktwal na character tulad ng caret symbol, dollar sign, o period.

Maaari nating ipahiwatig na gusto lang nating tumugma sa character sa pamamagitan ng pag-prefix ng character na iyon ng backslash. Halimbawa, maaari nating hanapin ang money amounts gamit ang sumusunod na regular expression.

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

Dahil nag-prefix tayo ng dollar sign ng backslash, aktwal na tumutugma ito sa dollar sign sa input string sa halip na tumugma sa “dulo ng linya”, at ang natitirang parte ng regular expression ay tumutugma sa isa o higit pang digits o period

character. Tandaan, tulad ng nakita natin sa itaas, sa loob ng square brackets, ang characters ay hindi "espesyal". Kaya kapag sinasabi natin ang `[0-9.]`, nangangahulugan ito ng digits o period. Sa labas ng square brackets, ang period ay ang "wild-card" character at tumutugma sa anumang character. Sa loob ng square brackets, ang period ay period.

11.5 Summary

Habang ito ay sumasayad lang sa ibabaw ng regular expressions, natutunan natin ang kaunti tungkol sa language ng regular expressions. Sila ay search strings na may espesyal na characters sa kanila na nagko-communicate ng iyong mga nais sa regular expression system tungkol sa kung ano ang nagde-define ng "matching" at kung ano ang kinukuha mula sa matched strings. Narito ang ilan sa mga espesyal na characters at character sequences:

`^` Tumutugma sa simula ng linya.

`$` Tumutugma sa dulo ng linya.

`.` Tumutugma sa anumang character (wildcard).

`\s` Tumutugma sa whitespace character.

`\S` Tumutugma sa non-whitespace character (kabaligtaran ng \s).

`*` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma zero o higit pang beses.

`*?` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma zero o higit pang beses sa "non-greedy mode".

`+` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma isa o higit pang beses.

`+?` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma isa o higit pang beses sa "non-greedy mode".

`?` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma zero o isang beses.

`??` Nalalapat sa agad na nauunang character(s) at nagpapahiwatig na tumugma zero o isang beses sa "non-greedy mode".

`[aeiou]` Tumutugma sa isang character hangga't ang character na iyon ay nasa tinukoy na set. Sa halimbawang ito, tutugma ito sa "a", "e", "i", "o", o "u", pero walang iba pang characters.

`[a-z0-9]` Maaari mong tukuyin ang ranges ng characters gamit ang minus sign. Ang halimbawang ito ay isang character na dapat lowercase letter o digit.

`[^A-Za-z]` Kapag ang unang character sa set notation ay caret, ito ay binabaligtad ang logic. Ang halimbawang ito ay tumutugma sa isang character na anumang bagay *maliban sa* uppercase o lowercase letter.

`( )` Kapag ang parentheses ay idinagdag sa regular expression, hindi sila pinapansin para sa layunin ng matching, pero nagpapahintulot sa iyo na kunin ang

partikular na subset ng matched string sa halip na buong string kapag gumagamit ng `findall()`.

`\b` Tumutugma sa empty string, pero lang sa simula o dulo ng salita.

`\B` Tumutugma sa empty string, pero hindi sa simula o dulo ng salita.

`\d` Tumutugma sa anumang decimal digit; katumbas ng set [0-9].

`\D` Tumutugma sa anumang non-digit character; katumbas ng set [^0-9].

11.6 Bonus section for Unix / Linux users

Ang suporta para sa paghahanap ng files gamit ang regular expressions ay naka-build sa Unix operating system mula noong 1960s at available ito sa halos lahat ng programming languages sa isang form o iba pa.

Sa katunayan, mayroong command-line program na naka-build sa Unix na tinatawag na *grep* (Generalized Regular Expression Parser) na gumagawa ng halos pareho sa `search()` examples sa chapter na ito. Kaya kung mayroon kang Macintosh o Linux system, maaari mong subukan ang sumusunod na commands sa command-line window mo.

```
$ grep '^From:' mbox-short.txt
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

Sinasabi nito sa `grep` na ipakita sa iyo ang lines na nagsisimula sa string "From:" sa file na *mbox-short.txt*. Kung mag-eksperimento ka sa `grep` command ng kaunti at basahin ang documentation para sa `grep`, makikita mo ang ilang subtle differences sa pagitan ng regular expression support sa Python at ang regular expression support sa `grep`. Bilang halimbawa, ang `grep` ay hindi sumusuporta sa non-blank character na `\S` kaya kailangan mong gamitin ang medyo mas kumplikadong set notation na `[^ ]`, na simpleng nangangahulugang tumugma sa character na anumang bagay maliban sa space.

11.7 Debugging

Ang Python ay may ilang simple at rudimentary built-in documentation na maaaring maging kapaki-pakinabang kung kailangan mo ng mabilis na refresher para matrigger ang memory mo tungkol sa eksaktong pangalan ng partikular na method. Ang documentation na ito ay maaaring tingnan sa Python interpreter sa interactive mode.

Maaari mong buksan ang interactive help system gamit ang `help()`.

```
>>> help()

help> modules
```

Kung alam mo kung anong module ang gusto mong gamitin, maaari mong gamitin ang `dir()` command para hanapin ang methods sa module tulad ng sumusunod:

```
>>> import re
>>> dir(re)
[.. 'compile', 'copy_reg', 'error', 'escape', 'findall',
'finditer', 'match', 'purge', 'search', 'split', 'sre_compile',
'sre_parse', 'sub', 'subn', 'sys', 'template']
```

Maaari mo ring makuha ang kaunting documentation tungkol sa partikular na method gamit ang help command na pinagsama sa gustong method.

```
>>> help (re.search)
Help on function search in module re:

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a match object, or None if no match was found.
>>>
```

Ang built-in documentation ay hindi masyadong malawak, pero maaari itong maging kapaki-pakinabang kapag nagmamadali ka o walang access sa web browser o search engine.

11.8 Glossary

brittle code Code na gumagana kapag ang input data ay nasa partikular na format pero madaling masira kung may deviation mula sa tamang format. Tinatawag natin itong "brittle code" dahil madali itong masira.

greedy matching Ang konsepto na ang `+` at `*` characters sa regular expression ay lumalawak palabas para tumugma sa pinakamalaking posibleng string.

grep Command na available sa karamihan ng Unix systems na naghahanap sa text files na naghahanap ng lines na tumutugma sa regular expressions. Ang command name ay nangangahulugang "Generalized Regular Expression Parser".

regular expression Language para ipahayag ang mas kumplikadong search strings. Ang regular expression ay maaaring maglalaman ng espesyal na characters na nagpapahiwatig na ang search ay tumutugma lang sa simula o dulo ng linya o marami pang iba na katulad na capabilities.

wild card Espesyal na character na tumutugma sa anumang character. Sa regular expressions ang wild-card character ay ang period.

11.9 Exercises

Exercise 1: Sumulat ng simpleng program para i-simulate ang operation ng `grep` command sa Unix. Magtanong sa user na mag-enter ng regular expression at bilangin ang bilang ng lines na tumugma sa regular expression:

```
$ python grep.py
Enter a regular expression: ^Author
mbox.txt had 1798 lines that matched ^Author

$ python grep.py
Enter a regular expression: ^X-
mbox.txt had 14368 lines that matched ^X-

$ python grep.py
Enter a regular expression: java$
mbox.txt had 4175 lines that matched java$
```

Exercise 2: Sumulat ng program para hanapin ang lines ng form:

```
New Revision: 39772
```

Kunin ang numero mula sa bawat isa sa lines gamit ang regular expression at ang `findall()` method. I-compute ang average ng mga numero at mag-print ng average bilang integer.

```
Enter file:mbox.txt
38549

Enter file:mbox-short.txt
39756
```

Chapter 12

Networked programs

Habang marami sa mga halimbawa sa libro na ito ay nakatuon sa pagbabasa ng files at paghahanap ng data sa mga files na iyon, mayroong maraming iba't ibang sources ng impormasyon kapag isinasaalang-alang ng isa ang Internet.

Sa chapter na ito magpapanggap tayo bilang web browser at kukuha ng web pages gamit ang Hypertext Transfer Protocol (HTTP). Pagkatapos magbabasa tayo sa web page data at i-parse ito.

12.1 Hypertext Transfer Protocol - HTTP

Ang network protocol na nagpapagana sa web ay talagang medyo simple at may built-in support sa Python na tinatawag na `socket` na nagpapadali sa paggawa ng network connections at pagkuha ng data sa pamamagitan ng mga sockets na iyon sa Python program.

Ang *socket* ay halos katulad ng file, maliban sa isang socket ay nagbibigay ng two-way connection sa pagitan ng dalawang programs. Maaari mong pareho magbasa mula at sumulat sa parehong socket. Kung sumusulat ka ng isang bagay sa socket, ipinapadala ito sa application sa kabilang dulo ng socket. Kung nagbabasa ka mula sa socket, binibigyan ka ng data na ipinadala ng iba pang application.

Pero kung susubukan mong magbasa ng socket[1] kapag ang program sa kabilang dulo ng socket ay hindi pa nagpadala ng anumang data, nakaupo ka lang at naghihintay. Kung ang programs sa parehong dulo ng socket ay simpleng naghihintay ng data nang hindi nagpapadala ng anumang bagay, maghihintay sila ng napakatagal, kaya mahalagang parte ng programs na nakikipag-communicate sa Internet ay magkaroon ng ilang uri ng protocol.

Ang protocol ay set ng tumpak na rules na nagde-determine kung sino ang uunang pumunta, kung ano ang gagawin nila, at pagkatapos kung ano ang mga responses sa mensaheng iyon, at sino ang susunod na magpapadala, at iba pa. Sa isang diwa ang dalawang applications sa magkabilang dulo ng socket ay gumagawa ng sayaw at sinisiguro na hindi tumapak sa paa ng isa't isa.

[1] Kung gusto mong matuto pa tungkol sa sockets, protocols o kung paano nabubuo ang web servers, maaari mong i-explore ang course sa https://www.dj4e.com.

Mayroong maraming dokumento na naglalarawan sa mga network protocols na ito. Ang Hypertext Transfer Protocol ay inilarawan sa sumusunod na dokumento:

https://www.w3.org/Protocols/rfc2616/rfc2616.txt

Ito ay mahabang at kumplikadong 176-page na dokumento na may maraming detalye. Kung nakakainteresante ito sa iyo, huwag mag-atubiling basahin ang lahat. Pero kung titingnan mo ang page 36 ng RFC2616 makikita mo ang syntax para sa GET request. Para humingi ng dokumento mula sa web server, gumagawa tayo ng connection, hal. sa `www.pr4e.org` server sa port 80, at pagkatapos nagpapadala ng linya ng form

```
GET http://data.pr4e.org/romeo.txt HTTP/1.0
```

kung saan ang pangalawang parameter ay ang web page na hinihingi natin, at pagkatapos nagpapadala din tayo ng blank line. Ang web server ay magre-respond ng ilang header impormasyon tungkol sa dokumento at blank line na sinusundan ng dokumento content.

12.2 The world's simplest web browser

Marahil ang pinakamadaling paraan para ipakita kung paano gumagana ang HTTP protocol ay sumulat ng napakasimpleng Python program na gumagawa ng connection sa web server at sumusunod sa rules ng HTTP protocol para humingi ng dokumento at ipakita ang ipinadala ng server.

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if len(data) < 1:
        break
    print(data.decode(),end='')

mysock.close()

# Code: https://www.py4e.com/code3/socket1.py
```

Una ang program ay gumagawa ng connection sa port 80 sa server www.pr4e.com. Dahil ang program natin ay gumaganap bilang "web browser", ang HTTP protocol ay nagsasabi na dapat nating ipadala ang GET command na sinusundan ng blank line. Ang `\r\n` ay nangangahulugang EOL (end of line), kaya ang `\r\n\r\n` ay nangangahulugang walang anuman sa pagitan ng dalawang EOL sequences. Iyon ang katumbas ng blank line.

Kapag naipadala na natin ang blank line na iyon, sumusulat tayo ng loop na tumatanggap ng data sa 512-character chunks mula sa socket at nagpi-print ng

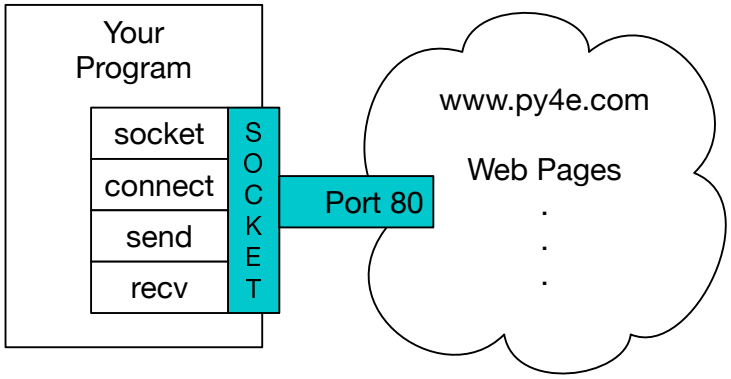

Figure 12.1: A Socket Connection

data hanggang wala nang data na mababasa (i.e., ang recv() ay nagre-return ng empty string).

Ang program ay gumagawa ng sumusunod na output:

```
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 18:52:55 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Sat, 13 May 2017 11:22:22 GMT
ETag: "a7-54f6609245537"
Accept-Ranges: bytes
Content-Length: 167
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: text/plain

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
```

Ang output ay nagsisimula sa headers na ipinapadala ng web server para ilarawan ang dokumento. Halimbawa, ang `Content-Type` header ay nagpapahiwatig na ang dokumento ay plain text document (`text/plain`).

Pagkatapos ipadala sa atin ng server ang headers, nagdaragdag ito ng blank line para ipahiwatig ang dulo ng headers, at pagkatapos nagpapadala ng aktwal na data ng file *romeo.txt*.

Ang halimbawang ito ay nagpapakita kung paano gumawa ng low-level network connection gamit ang sockets. Ang sockets ay maaaring gamitin para makipag-communicate sa web server o sa mail server o marami pang ibang uri ng servers. Ang kailangan lang ay hanapin ang dokumento na naglalarawan sa protocol at sumulat ng code para magpadala at tumanggap ng data ayon sa protocol.

Gayunpaman, dahil ang protocol na pinakakaraniwang ginagamit natin ay ang HTTP web protocol, ang Python ay may espesyal na library na partikular na

idinisenyo para suportahan ang HTTP protocol para sa pagkuha ng mga dokumento at data sa web.

Isa sa mga requirements para gamitin ang HTTP protocol ay ang pangangailangan na magpadala at tumanggap ng data bilang bytes objects, sa halip na strings. Sa naunang halimbawa, ang `encode()` at `decode()` methods ay nagko-convert ng strings sa bytes objects at pabalik.

Ang susunod na halimbawa ay gumagamit ng `b''` notation para tukuyin na ang variable ay dapat i-store bilang bytes object. Ang `encode()` at `b''` ay katumbas.

```
>>> b'Hello world'
b'Hello world'
>>> 'Hello world'.encode()
b'Hello world'
```

12.3 Retrieving an image over HTTP

Sa halimbawa sa itaas, kumuha tayo ng plain text file na may newlines sa file at simpleng kinopya ang data sa screen habang tumatakbo ang program. Maaari tayong gumamit ng katulad na program para kumuha ng image gamit ang HTTP. Sa halip na kopyahin ang data sa screen habang tumatakbo ang program, ina-accumulate natin ang data sa string, tinatanggal ang headers, at pagkatapos sinasave ang image data sa file tulad ng sumusunod:

```
import socket
import time

HOST = 'data.pr4e.org'
PORT = 80
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))
mysock.sendall(b'GET http://data.pr4e.org/cover3.jpg HTTP/1.0\r\n\r\n')
count = 0
picture = b""

while True:
    data = mysock.recv(5120)
    if len(data) < 1: break
    #time.sleep(0.25)
    count = count + len(data)
    print(len(data), count)
    picture = picture + data

mysock.close()

# Look for the end of the header (2 CRLF)
pos = picture.find(b"\r\n\r\n")
print('Header length', pos)
print(picture[:pos].decode())
```

```
# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg", "wb")
fhand.write(picture)
fhand.close()

# Code: https://www.py4e.com/code3/urljpeg.py
```

Kapag tumatakbo ang program, gumagawa ito ng sumusunod na output:

```
$ python urljpeg.py
5120 5120
5120 10240
4240 14480
5120 19600
...
5120 214000
3200 217200
5120 222320
5120 227440
3167 230607
Header length 393
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 18:54:09 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Mon, 15 May 2017 12:27:40 GMT
ETag: "38342-54f8f2e5b6277"
Accept-Ranges: bytes
Content-Length: 230210
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: image/jpeg
```

Makikita mo na para sa url na ito, ang `Content-Type` header ay nagpapahiwatig na ang body ng dokumento ay image (`image/jpeg`). Kapag natapos na ang program, maaari mong tingnan ang image data sa pamamagitan ng pagbubukas ng file na `stuff.jpg` sa image viewer.

Habang tumatakbo ang program, makikita mo na hindi tayo nakakakuha ng 5120 characters sa bawat pagkakataon na tinatawag natin ang `recv()` method. Nakakakuha tayo ng kasing dami ng characters na na-transfer sa network sa atin ng web server sa sandaling tinatawag natin ang `recv()`. Sa halimbawang ito, nakakakuha tayo ng kasing kaunti ng 3200 characters sa bawat pagkakataon na humihingi tayo ng hanggang 5120 characters ng data.

Ang iyong results ay maaaring magkaiba depende sa network speed mo. Tandaan din na sa huling tawag sa `recv()` nakakakuha tayo ng 3167 bytes, na siyang dulo ng stream, at sa susunod na tawag sa `recv()` nakakakuha tayo ng zero-length string na nagsasabi sa atin na ang server ay tumawag na sa `close()` sa dulo nito ng socket at wala nang data na darating.

Maaari nating pabagalin ang sunud-sunod na `recv()` calls natin sa pamamagitan ng pag-uncomment sa tawag sa `time.sleep()`. Sa ganitong paraan, naghihintay tayo ng quarter ng segundo pagkatapos ng bawat tawag para makakuha ng "lead" ang server at magpadala ng mas maraming data sa atin bago tayo tumawag ulit sa `recv()`. Sa delay, sa lugar ang program ay nag-e-execute tulad ng sumusunod:

```
$ python urljpeg.py
5120 5120
5120 10240
5120 15360
...
5120 225280
5120 230400
207 230607
Header length 393
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 21:42:08 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Mon, 15 May 2017 12:27:40 GMT
ETag: "38342-54f8f2e5b6277"
Accept-Ranges: bytes
Content-Length: 230210
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Connection: close
Content-Type: image/jpeg
```

Ngayon maliban sa una at huling tawag sa `recv()`, nakakakuha na tayo ng 5120 characters sa bawat pagkakataon na humihingi tayo ng bagong data.

Mayroong buffer sa pagitan ng server na gumagawa ng `send()` requests at application natin na gumagawa ng `recv()` requests. Kapag pinatakbo natin ang program na may delay sa lugar, sa ilang punto ang server ay maaaring mapuno ang buffer sa socket at mapilit na mag-pause hanggang magsimulang mag-emptying ng buffer ang program natin. Ang pag-pause ng alinman sa sending application o receiving application ay tinatawag na "flow control."

12.4 Retrieving web pages with `urllib`

Habang maaari nating manu-manong magpadala at tumanggap ng data sa HTTP gamit ang socket library, mayroong mas simpleng paraan para gawin ang karaniwang gawaing ito sa Python sa pamamagitan ng paggamit ng `urllib` library.

Gamit ang `urllib`, maaari mong tratuhin ang web page na parang file. Simpleng ipinapahiwatig mo lang kung aling web page ang gusto mong kunin at ang `urllib` ay nagha-handle ng lahat ng HTTP protocol at header details.

Ang katumbas na code para basahin ang *romeo.txt* file mula sa web gamit ang `urllib` ay ganito:

```
import urllib.request

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())

# Code: https://www.py4e.com/code3/urllib1.py
```

Kapag nabuksan na ang web page gamit ang `urllib.request.urlopen`, maaari nating tratuhin ito na parang file at basahin ito gamit ang `for` loop.

Kapag tumatakbo ang program, nakikita lang natin ang output ng contents ng file. Ang headers ay ipinapadala pa rin, pero ang `urllib` code ay kumukonsumo ng headers at nagre-return lang ng data sa atin.

```
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
```

Bilang halimbawa, maaari tayong sumulat ng program para kunin ang data para sa `romeo.txt` at i-compute ang frequency ng bawat salita sa file tulad ng sumusunod:

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')

counts = dict()
for line in fhand:
    words = line.decode().split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1
print(counts)

# Code: https://www.py4e.com/code3/urlwords.py
```

Muli, kapag nabuksan na natin ang web page, maaari nating basahin ito na parang local file.

12.5 Reading binary files using `urllib`

Minsan gusto mong kunin ang non-text (o binary) file tulad ng image o video file. Ang data sa mga files na ito ay karaniwang hindi kapaki-pakinabang na i-print, pero madali mong magagawa ng kopya ng URL sa local file sa hard disk mo gamit ang `urllib`.

Ang pattern ay buksan ang URL at gamitin ang `read` para i-download ang buong contents ng dokumento sa string variable (`img`) pagkatapos isulat ang impormasyong iyon sa local file tulad ng sumusunod:

```
import urllib.request, urllib.parse, urllib.error

img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()

# Code: https://www.py4e.com/code3/curl1.py
```

Ang program na ito ay nagbabasa ng lahat ng data nang sabay-sabay sa network at nag-i-store nito sa variable na `img` sa main memory ng iyong computer, pagkatapos binubuksan ang file na `cover.jpg` at isinusulat ang data sa disk mo. Ang `wb` argument para sa `open()` ay nagbubukas ng binary file para sa pagsusulat lang. Ang program na ito ay gagana kung ang laki ng file ay mas maliit kaysa sa laki ng memory ng computer mo.

Gayunpaman kung ito ay malaking audio o video file, ang program na ito ay maaaring mag-crash o hindi bababa ay tumakbo nang napakabagal kapag naubusan ng memory ang computer mo. Para maiwasan ang pagkaubos ng memory, kinukuha natin ang data sa blocks (o buffers) at pagkatapos isinusulat ang bawat block sa disk mo bago kunin ang susunod na block. Sa ganitong paraan ang program ay maaaring magbasa ng anumang laking file nang hindi ginagamit ang lahat ng memory na mayroon ka sa computer mo.

```
import urllib.request, urllib.parse, urllib.error

img = urllib.request.urlopen('http://data.pr4e.org/cover3.jpg')
fhand = open('cover3.jpg', 'wb')
size = 0
while True:
    info = img.read(100000)
    if len(info) < 1: break
    size = size + len(info)
    fhand.write(info)

print(size, 'characters copied.')
fhand.close()

# Code: https://www.py4e.com/code3/curl2.py
```

Sa halimbawang ito, nagbabasa lang tayo ng 100,000 characters sa isang pagkakataon at pagkatapos isinusulat ang mga characters na iyon sa file na `cover3.jpg` bago kunin ang susunod na 100,000 characters ng data mula sa web.

Ang program na ito ay tumatakbo tulad ng sumusunod:

```
python curl2.py
230210 characters copied.
```

12.6 Parsing HTML and scraping the web

Isa sa karaniwang gamit ng `urllib` capability sa Python ay *i-scrape* ang web. Ang web scraping ay kapag sumusulat tayo ng program na nagpapanggap bilang web browser at kumukuha ng pages, pagkatapos sinusuri ang data sa mga pages na iyon na naghahanap ng patterns.

Bilang halimbawa, ang search engine tulad ng Google ay titingnan ang source ng isang web page at kukunin ang links sa iba pang pages at kukunin ang mga pages na iyon, kumukuha ng links, at iba pa. Gamit ang technique na ito, ang Google ay *nag-spider* sa halos lahat ng pages sa web.

Gumagamit din ang Google ng frequency ng links mula sa pages na nakikita nito patungo sa partikular na page bilang isang sukat kung gaano "important" ang page at kung gaano kataas dapat lumabas ang page sa search results nito.

12.7 Parsing HTML using regular expressions

Ang isang simpleng paraan para mag-parse ng HTML ay gumamit ng regular expressions para paulit-ulit na maghanap at kunin ang substrings na tumutugma sa partikular na pattern.

Narito ang simpleng web page:

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

Maaari tayong gumawa ng well-formed regular expression para tumugma at kunin ang link values mula sa text sa itaas tulad ng sumusunod:

```
href="http[s]?://.+?"
```

Ang regular expression natin ay naghahanap ng strings na nagsisimula sa "href="http://" o "href="https://", na sinusundan ng isa o higit pang characters (`.+?`), na sinusundan ng isa pang double quote. Ang question mark sa likod ng `[s]?` ay nagpapahiwatig na maghanap ng string na "http" na sinusundan ng zero o isang "s".

Ang question mark na idinagdag sa `.+?` ay nagpapahiwatig na ang match ay dapat gawin sa "non-greedy" fashion sa halip na "greedy" fashion. Ang non-greedy match ay sinusubukang hanapin ang *pinakamaliit* na posibleng matching string at ang greedy match ay sinusubukang hanapin ang *pinakamalaki* na posibleng matching string.

Nagdaragdag tayo ng parentheses sa regular expression natin para ipahiwatig kung aling parte ng matched string natin ang gusto nating kunin, at gumagawa ng sumusunod na program:

```
# Search for link values within URL input
import urllib.request, urllib.parse, urllib.error
import re
import ssl

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
links = re.findall(b'href="(http[s]?://.*?)"', html)
for link in links:
    print(link.decode())

# Code: https://www.py4e.com/code3/urlregex.py
```

Ang `ssl` library ay nagpapahintulot sa program na ito na ma-access ang web sites na mahigpit na nagpapatupad ng HTTPS. Ang `read` method ay nagre-return ng HTML source code bilang bytes object sa halip na mag-return ng HTTPResponse object. Ang `findall` regular expression method ay magbibigay sa atin ng list ng lahat ng strings na tumutugma sa aming regular expression, na nagre-return lang ng link text sa pagitan ng double quotes.

Kapag pinatakbo natin ang program at nag-input ng URL, makukuha natin ang sumusunod na output:

```
Enter - https://docs.python.org
https://docs.python.org/3/index.html
https://www.python.org/
https://docs.python.org/3.8/
https://docs.python.org/3.7/
https://docs.python.org/3.5/
https://docs.python.org/2.7/
https://www.python.org/doc/versions/
https://www.python.org/dev/peps/
https://wiki.python.org/moin/BeginnersGuide
https://wiki.python.org/moin/PythonBooks
https://www.python.org/doc/av/
https://www.python.org/
https://www.python.org/psf/donations/
http://sphinx.pocoo.org/
```

Ang regular expressions ay gumagana nang napakaganda kapag ang HTML mo ay well formatted at predictable. Pero dahil mayroong maraming “broken” HTML pages doon, ang solusyon na gumagamit lang ng regular expressions ay maaaring makaligtaan ang ilang valid links o magtapos sa masamang data.

Maaari itong malutas sa pamamagitan ng paggamit ng robust HTML parsing library.

12.8 Parsing HTML using BeautifulSoup

Kahit na ang HTML ay mukhang XML[2] at ang ilang pages ay maingat na ginawa para maging XML, karamihan ng HTML ay karaniwang sira sa mga paraan na nagdudulot na ang XML parser ay tumanggi sa buong page ng HTML bilang hindi wastong format.

Mayroong ilang Python libraries na maaaring tumulong sa iyo na mag-parse ng HTML at kumuha ng data mula sa pages. Ang bawat library ay may sariling strengths at weaknesses at maaari kang pumili ng isa batay sa iyong pangangailangan.

Bilang halimbawa, simpleng magpa-parse tayo ng ilang HTML input at kumuha ng links gamit ang *BeautifulSoup* library. Ang BeautifulSoup ay nagpaparaya sa lubhang flawed HTML at nagpapahintulot pa rin sa iyo na madaling kunin ang data na kailangan mo. Maaari mong i-download at i-install ang BeautifulSoup code mula sa:

https://pypi.python.org/pypi/beautifulsoup4

Ang impormasyon tungkol sa pag-install ng BeautifulSoup gamit ang Python Package Index tool na `pip` ay available sa:

https://packaging.python.org/tutorials/installing-packages/

Gagamitin natin ang `urllib` para basahin ang page at pagkatapos gamitin ang `BeautifulSoup` para kunin ang `href` attributes mula sa anchor (`a`) tags.

```
# To run this, download the BeautifulSoup zip file
# http://www.py4e.com/code3/bs4.zip
# or pip install beautifulsoup4 to ensure you have the latest version
# and unzip it in the same directory as this file

import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl # defauts to certicate verification and most secure protocol (now

# Ignore SSL/TLS certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))

# Code: https://www.py4e.com/code3/urllinks.py
```

[2]Ang XML format ay inilarawan sa susunod na chapter.

Ang program ay nagpo-prompt para sa web address, pagkatapos binubuksan ang web page, binabasa ang data at ipinapasa ang data sa BeautifulSoup parser, at pagkatapos kumukuha ng lahat ng anchor tags at nagpi-print ng `href` attribute para sa bawat tag.

Kapag tumatakbo ang program, gumagawa ito ng sumusunod na output:

```
Enter - https://docs.python.org
genindex.html
py-modindex.html
https://www.python.org/
#
whatsnew/3.6.html
whatsnew/index.html
tutorial/index.html
library/index.html
reference/index.html
using/index.html
howto/index.html
installing/index.html
distributing/index.html
extending/index.html
c-api/index.html
faq/index.html
py-modindex.html
genindex.html
glossary.html
search.html
contents.html
bugs.html
about.html
license.html
copyright.html
download.html
https://docs.python.org/3.8/
https://docs.python.org/3.7/
https://docs.python.org/3.5/
https://docs.python.org/2.7/
https://www.python.org/doc/versions/
https://www.python.org/dev/peps/
https://wiki.python.org/moin/BeginnersGuide
https://wiki.python.org/moin/PythonBooks
https://www.python.org/doc/av/
genindex.html
py-modindex.html
https://www.python.org/
#
copyright.html
https://www.python.org/psf/donations/
bugs.html
http://sphinx.pocoo.org/
```

Ang list na ito ay mas mahaba dahil ang ilang HTML anchor tags ay relative paths (hal., tutorial/index.html) o in-page references (hal., '#') na hindi kasama ang "http://" o "https://", na siyang requirement sa regular expression natin.

Maaari mo ring gamitin ang BeautifulSoup para kunin ang iba't ibang parte ng bawat tag:

```
# To run this, download the BeautifulSoup zip file
# http://www.py4e.com/code3/bs4.zip
# and unzip it in the same directory as this file

from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, "html.parser")

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    # Look at the parts of a tag
    print('TAG:', tag)
    print('URL:', tag.get('href', None))
    print('Contents:', tag.contents[0])
    print('Attrs:', tag.attrs)

# Code: https://www.py4e.com/code3/urllink2.py
```

```
python urllink2.py
Enter - http://www.dr-chuck.com/page1.htm
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: ['\nSecond Page']
Attrs: [('href', 'http://www.dr-chuck.com/page2.htm')]
```

Ang `html.parser` ay ang HTML parser na kasama sa standard Python 3 library. Ang impormasyon tungkol sa iba pang HTML parsers ay available sa:

http://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

Ang mga halimbawang ito ay nagsisimula lang na ipakita ang kapangyarihan ng BeautifulSoup pagdating sa pag-parse ng HTML.

12.9 Bonus section for Unix / Linux users

Kung mayroon kang Linux, Unix, o Macintosh computer, malamang mayroon kang commands na naka-build sa operating system mo na kumukuha ng parehong plain

text at binary files gamit ang HTTP o File Transfer (FTP) protocols. Isa sa mga commands na ito ay `curl`:

```
$ curl -O http://www.py4e.com/cover.jpg
```

Ang command na `curl` ay maikli para sa "copy URL" at kaya ang dalawang halimbawa na nakalista kanina para kumuha ng binary files gamit ang `urllib` ay matalinong pinangalanan na `curl1.py` at `curl2.py` sa www.py4e.com/code3 dahil nag-i-implement sila ng katulad na functionality sa `curl` command. Mayroon ding `curl3.py` sample program na gumagawa ng gawaing ito nang mas epektibo, kung sakaling gusto mong talagang gamitin ang pattern na ito sa program na sinusulat mo.

Ang pangalawang command na gumagana nang katulad ay `wget`:

```
$ wget http://www.py4e.com/cover.jpg
```

Pareho sa mga commands na ito ay nagpapasimple sa pagkuha ng webpages at remote files.

12.10 Glossary

BeautifulSoup Python library para sa pag-parse ng HTML documents at pagkuha ng data mula sa HTML documents na nagko-compensate para sa karamihan ng imperfections sa HTML na karaniwang hindi pinapansin ng browsers. Maaari mong i-download ang BeautifulSoup code mula sa www.crummy.com.

port Numero na karaniwang nagpapahiwatig kung aling application ang kinokontak mo kapag gumagawa ka ng socket connection sa server. Bilang halimbawa, ang web traffic ay karaniwang gumagamit ng port 80 habang ang email traffic ay gumagamit ng port 25.

scrape Kapag ang program ay nagpapanggap bilang web browser at kumukuha ng web page, pagkatapos tumitingin sa web page content. Kadalasan ang mga programs ay sumusunod sa links sa isang page para hanapin ang susunod na page para maaari nilang dumaan sa network ng pages o social network.

socket Network connection sa pagitan ng dalawang applications kung saan ang applications ay maaaring magpadala at tumanggap ng data sa alinmang direksyon.

spider Ang gawain ng web search engine na kumukuha ng page at pagkatapos lahat ng pages na naka-link mula sa page at iba pa hanggang mayroon na silang halos lahat ng pages sa Internet na ginagamit nila para gumawa ng search index nila.

12.11 Exercises

Exercise 1: Baguhin ang socket program na `socket1.py` para mag-prompt sa user para sa URL para makabasa ito ng anumang web page.

Maaari mong gamitin ang `split('/')` para hatiin ang URL sa component parts nito para makakuha ka ng host name para sa socket `connect` call. Magdagdag ng error checking gamit ang `try` at `except` para ma-handle ang kondisyon kung saan ang user ay nag-e-enter ng hindi wastong format o hindi umiiral na URL.

Exercise 2: Baguhin ang socket program mo para bilangin ang bilang ng characters na natanggap nito at huminto sa pag-display ng anumang text pagkatapos na ipakita ang 3000 characters. Ang program ay dapat kumuha ng buong dokumento at bilangin ang kabuuang bilang ng characters at ipakita ang bilang ng bilang ng characters sa dulo ng dokumento.

Exercise 3: Gumamit ng `urllib` para i-replicate ang naunang exercise ng (1) pagkuha ng dokumento mula sa URL, (2) pag-display ng hanggang 3000 characters, at (3) pagbibilang ng kabuuang bilang ng characters sa dokumento. Huwag mag-alala tungkol sa headers para sa exercise na ito, simpleng ipakita ang unang 3000 characters ng document contents.

Exercise 4: Baguhin ang `urllinks.py` program para kunin at bilangin ang paragraph (p) tags mula sa retrieved HTML document at ipakita ang bilang ng paragraphs bilang output ng program mo. Huwag ipakita ang paragraph text, bilangin lang sila. I-test ang program mo sa ilang maliliit na web pages pati na rin sa ilang mas malalaking web pages.

Exercise 5: (Advanced) Baguhin ang socket program para ipakita lang ang data pagkatapos matanggap ang headers at blank line. Tandaan na ang `recv` ay tumatanggap ng characters (newlines at lahat), hindi lines.

Chapter 13

Using Web Services

Nang naging madali na ang pagkuha ng documents at pag-parse ng documents sa HTTP gamit ang programs, hindi nagtagal bago nabuo ang approach kung saan nagsimula tayong gumawa ng documents na partikular na idinisenyo para konsumahin ng iba pang programs (i.e., hindi HTML para ipakita sa browser).

Mayroong dalawang karaniwang format na ginagamit natin kapag nagpapalitan ng data sa web. Ang eXtensible Markup Language (XML) ay ginagamit na ng napakatagal at pinakaangkop para sa pagpapalitan ng document-style data. Kapag ang programs ay gusto lang magpalitan ng dictionaries, lists, o iba pang internal impormasyon sa isa't isa, gumagamit sila ng JavaScript Object Notation (JSON) (tingnan ang www.json.org). Titingnan natin ang parehong format.

13.1 eXtensible Markup Language - XML

Ang XML ay mukhang katulad ng HTML, pero ang XML ay mas structured kaysa sa HTML. Narito ang sample ng XML document:

```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

Ang bawat pares ng opening (hal., `<person>`) at closing tags (hal., `</person>`) ay kumakatawan sa *element* o *node* na may parehong pangalan sa tag (hal., `person`). Ang bawat element ay maaaring may ilang text, ilang attributes (hal., `hide`), at iba pang nested elements. Kung ang XML element ay empty (i.e., walang content), maaari itong ilarawan ng self-closing tag (hal., `<email />`).

Kadalasan kapaki-pakinabang na isipin ang XML document bilang tree structure kung saan mayroong top element (dito: `person`), at ang iba pang tags (hal., `phone`) ay iginuhit bilang *children* ng kanilang *parent* elements.

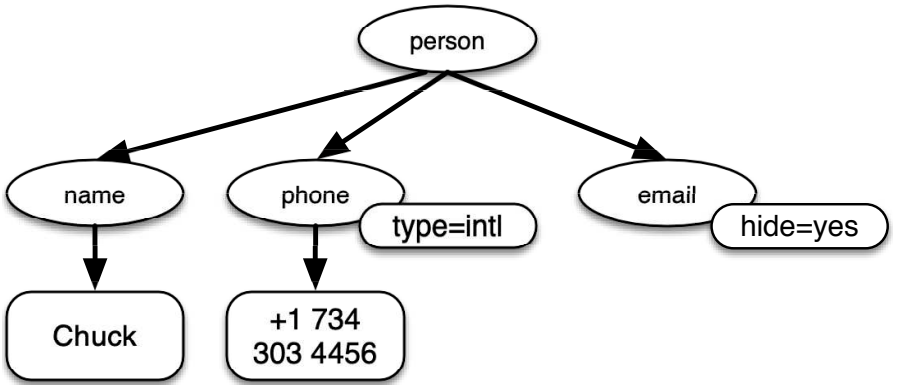

Figure 13.1: A Tree Representation of XML

13.2 Parsing XML

Narito ang simpleng application na nagpa-parse ng ilang XML at kumukuha ng ilang data elements mula sa XML:

```
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>'''

tree = ET.fromstring(data)
print('Name:', tree.find('name').text)
print('Attr:', tree.find('email').get('hide'))

# Code: https://www.py4e.com/code3/xml1.py
```

Ang triple single quote (`'''`), pati na rin ang triple double quote (`"""`), ay nagpapahintulot sa paggawa ng strings na sumasaklaw sa maraming linya.

Ang pagtawag sa `fromstring` ay nagko-convert ng string representation ng XML sa "tree" ng XML elements. Kapag ang XML ay nasa tree, mayroon tayong serye ng methods na maaari nating tawagin para kunin ang mga parte ng data mula sa XML string. Ang `find` function ay naghahanap sa XML tree at kumukuha ng element na tumutugma sa tinukoy na tag.

```
Name: Chuck
Attr: yes
```

Ang paggamit ng XML parser tulad ng `ElementTree` ay may advantage na habang ang XML sa halimbawang ito ay medyo simple, lumalabas na mayroong maraming rules tungkol sa valid XML, at ang paggamit ng `ElementTree` ay nagpapahintulot sa atin na kunin ang data mula sa XML nang hindi nag-aalala tungkol sa rules ng XML syntax.

13.3 Looping through nodes

Kadalasan ang XML ay may maraming nodes at kailangan nating sumulat ng loop para iproseso ang lahat ng nodes. Sa sumusunod na program, naglo-loop tayo sa lahat ng `user` nodes:

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <users>
    <user x="2">
      <id>001</id>
      <name>Chuck</name>
    </user>
    <user x="7">
      <id>009</id>
      <name>Brent</name>
    </user>
  </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get('x'))

# Code: https://www.py4e.com/code3/xml2.py
```

Ang `findall` method ay kumukuha ng Python list ng subtrees na kumakatawan sa `user` structures sa XML tree. Pagkatapos maaari tayong sumulat ng `for` loop na tumitingin sa bawat user node, at nagpi-print ng `name` at `id` text elements pati na rin ang `x` attribute mula sa `user` node.

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
```

```
Id 009
Attribute 7
```

Mahalagang isama ang lahat ng parent level elements sa `findall` statement maliban sa top level element (hal., `users/user`). Kung hindi, ang Python ay hindi makakahanap ng anumang gustong nodes.

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <users>
    <user x="2">
      <id>001</id>
      <name>Chuck</name>
    </user>
    <user x="7">
      <id>009</id>
      <name>Brent</name>
    </user>
  </users>
</stuff>'''

stuff = ET.fromstring(input)

lst = stuff.findall('users/user')
print('User count:', len(lst))

lst2 = stuff.findall('user')
print('User count:', len(lst2))
```

Ang `lst` ay nag-i-store ng lahat ng `user` elements na nested sa loob ng kanilang `users` parent. Ang `lst2` ay naghahanap ng `user` elements na hindi nested sa loob ng top level `stuff` element kung saan walang anuman.

```
User count: 2
User count: 0
```

13.4 JavaScript Object Notation - JSON

Ang JSON format ay naging inspirasyon mula sa object at array format na ginagamit sa JavaScript language. Pero dahil ang Python ay naimbento bago ang JavaScript, ang syntax ng Python para sa dictionaries at lists ay naimpluwensyahan ang syntax ng JSON. Kaya ang format ng JSON ay halos kapareho ng kombinasyon ng Python lists at dictionaries.

Narito ang JSON encoding na halos katumbas ng simpleng XML mula sa itaas:

```
{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}
```

Mapapansin mo ang ilang pagkakaiba. Una, sa XML, maaari tayong magdagdag ng attributes tulad ng "intl" sa "phone" tag. Sa JSON, mayroon lang tayong key-value pairs. Gayundin ang XML "person" tag ay nawala, pinalitan ng set ng outer curly braces.

Sa pangkalahatan, ang JSON structures ay mas simple kaysa sa XML dahil ang JSON ay may mas kaunting capabilities kaysa sa XML. Pero ang JSON ay may advantage na nagma-map ito *directly* sa ilang kombinasyon ng dictionaries at lists. At dahil halos lahat ng programming languages ay may katumbas sa dictionaries at lists ng Python, ang JSON ay napakalikas na format para magkaroon ng dalawang cooperating programs na magpalitan ng data.

Ang JSON ay mabilis na naging format ng pagpipilian para sa halos lahat ng data exchange sa pagitan ng applications dahil sa relatibong simplicity nito kumpara sa XML.

13.5 Parsing JSON

Ginagawa natin ang JSON natin sa pamamagitan ng pag-nest ng dictionaries at lists ayon sa pangangailangan. Sa halimbawang ito, kinakatawan natin ang list ng users kung saan ang bawat user ay set ng key-value pairs (i.e., dictionary). Kaya mayroon tayong list ng dictionaries.

Sa sumusunod na program, ginagamit natin ang built-in `json` library para mag-parse ng JSON at magbasa sa data. I-compare ito nang mabuti sa katumbas na XML data at code sa itaas. Ang JSON ay may mas kaunting detalye, kaya dapat nating malaman nang maaga na nakakakuha tayo ng list at ang list ay ng users at ang bawat user ay set ng key-value pairs. Ang JSON ay mas succinct (advantage) pero mas kaunti rin ang self-describing (a disadvantage).

```
import json

data = '''
[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
```

```
    "x" : "7",
    "name" : "Brent"
  }
]'''

info = json.loads(data)
print('User count:', len(info))

for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])

# Code: https://www.py4e.com/code3/json2.py
```

Kung i-compare mo ang code para kunin ang data mula sa parsed JSON at XML makikita mo na ang nakukuha natin mula sa `json.loads()` ay Python list na dinadaanan natin gamit ang `for` loop, at ang bawat item sa loob ng list na iyon ay Python dictionary. Kapag na-parse na ang JSON, maaari nating gamitin ang Python index operator para kunin ang iba't ibang bits ng data para sa bawat user. Hindi natin kailangang gamitin ang JSON library para mag-dig sa parsed JSON, dahil ang returned data ay simpleng native Python structures.

Ang output ng program na ito ay eksaktong pareho sa XML version sa itaas.

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

Sa pangkalahatan, mayroong industry trend palayo sa XML at patungo sa JSON para sa web services. Dahil ang JSON ay mas simple at mas direktang nagma-map sa native data structures na mayroon na tayo sa programming languages, ang parsing at data extraction code ay karaniwang mas simple at mas direkta kapag gumagamit ng JSON. Pero ang XML ay mas self-descriptive kaysa sa JSON at kaya mayroong ilang applications kung saan ang XML ay nananatiling may advantage. Halimbawa, karamihan ng word processors ay nag-i-store ng documents internally gamit ang XML sa halip na JSON.

13.6 Application Programming Interfaces

Mayroon na tayong kakayahang magpalitan ng data sa pagitan ng applications gamit ang Hypertext Transport Protocol (HTTP) at paraan para kumatawan sa complex data na ipinapadala natin pabalik-balik sa pagitan ng mga applications na ito gamit ang eXtensible Markup Language (XML) o JavaScript Object Notation (JSON).

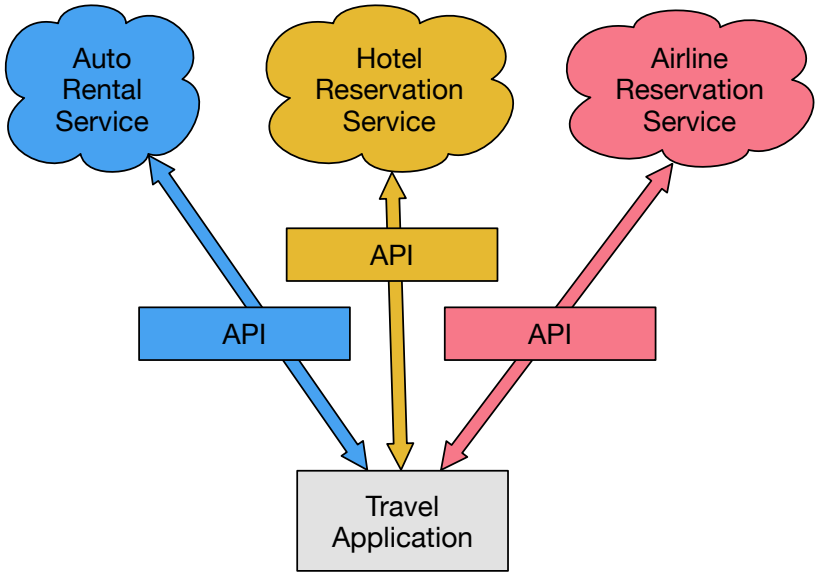

Figure 13.2: Service-oriented architecture

Ang susunod na hakbang ay simulan na tukuyin at dokumentado ang "contracts" sa pagitan ng applications gamit ang mga techniques na ito. Ang pangkalahatang pangalan para sa mga application-to-application contracts na ito ay *Application Program Interfaces* (APIs). Kapag gumagamit tayo ng API, karaniwang isang program ang gumagawa ng set ng *services* na available para gamitin ng iba pang applications at nagpu-publish ng APIs (i.e., ang "rules") na dapat sundin para ma-access ang services na ibinigay ng program.

Kapag nagsisimula na tayong gumawa ng programs natin kung saan ang functionality ng aming program ay kasama ang access sa services na ibinigay ng iba pang programs, tinatawag natin ang approach na *Service-oriented architecture* (SOA). Ang SOA approach ay isa kung saan ang overall application natin ay gumagamit ng services ng iba pang applications. Ang non-SOA approach ay kung saan ang application ay isang solong standalone application na naglalaman ng lahat ng code na kailangan para i-implement ang application.

Nakikita natin ang maraming halimbawa ng SOA kapag gumagamit tayo ng web. Maaari tayong pumunta sa isang web site at mag-book ng air travel, hotels, at automobiles lahat mula sa isang site. Ang data para sa hotels ay hindi naka-store sa airline computers. Sa halip, ang airline computers ay nakikipag-ugnayan sa services sa hotel computers at kumukuha ng hotel data at ipinakita ito sa user. Kapag sumang-ayon ang user na gumawa ng hotel reservation gamit ang airline site, ang airline site ay gumagamit ng iba pang web service sa hotel systems para talagang gawin ang reservation. At kapag oras na para singilin ang credit card mo para sa buong transaction, iba pang computers pa rin ang kasangkot sa proseso.

Ang Service-oriented architecture ay may maraming advantages, kasama ang: (1) palaging nagma-maintain lang tayo ng isang kopya ng data (ito ay partikular na mahalaga para sa mga bagay tulad ng hotel reservations kung saan ayaw nating mag-over-commit) at (2) ang mga may-ari ng data ay maaaring magtakda ng rules tungkol sa paggamit ng kanilang data. Sa mga advantages na ito, ang SOA system

ay dapat maingat na idinisenyo para magkaroon ng magandang performance at matugunan ang pangangailangan ng user.

Kapag ang application ay gumagawa ng set ng services sa API nito na available sa web, tinatawag natin ang mga ito na *web services*.

13.7 Security and API usage

Napakakaraniwan na kailangan mo ng API key para magamit ang API ng vendor. Ang pangkalahatang ideya ay gusto nilang malaman kung sino ang gumagamit ng services nila at kung gaano karami ang ginagamit ng bawat user. Marahil mayroon silang libre at bayad na tiers ng services nila o may policy na naglilimita sa bilang ng requests na maaaring gawin ng isang indibidwal sa partikular na panahon period.

Minsan kapag nakuha mo na ang API key mo, simpleng isama mo lang ang key bilang parte ng POST data o marahil bilang parameter sa URL kapag tinatawag ang API.

Sa ibang pagkakataon, ang vendor ay gusto ng mas mataas na assurance ng source ng requests at kaya inaasahan nila na magpadala ka ng cryptographically signed messages gamit ang shared keys at secrets. Ang napakakaraniwang teknolohiya na ginagamit para mag-sign ng requests sa Internet ay tinatawag na *OAuth*. Maaari kang magbasa pa tungkol sa OAuth protocol sa www.oauth.net.

Sa kabutihang palad mayroong ilang maginhawa at libreng OAuth libraries para maiwasan mong sumulat ng OAuth implementation mula sa simula sa pamamagitan ng pagbabasa ng specification. Ang mga libraries na ito ay may iba't ibang complexity at may iba't ibang antas ng richness. Ang OAuth web site ay may impormasyon tungkol sa iba't ibang OAuth libraries.

13.8 Glossary

API Application Program Interface - Contract sa pagitan ng applications na nagde-define ng patterns ng interaction sa pagitan ng dalawang application components.

ElementTree Built-in Python library na ginagamit para mag-parse ng XML data.

JSON JavaScript Object Notation - Format na nagpapahintulot sa markup ng structured data batay sa syntax ng JavaScript Objects.

SOA Service-Oriented Architecture - Kapag ang application ay gawa sa components na konektado sa network.

XML eXtensible Markup Language - Format na nagpapahintulot sa markup ng structured data.

Chapter 14

Object-oriented programming

14.1 Managing larger programs

Sa simula ng libro na ito, nakabuo tayo ng apat na basic programming patterns na ginagamit natin para gumawa ng programs:

- Sequential code
- Conditional code (if statements)
- Repetitive code (loops)
- Store and reuse (functions)

Sa mga susunod na chapters, nag-explore tayo ng simple variables pati na rin ng collection data structures tulad ng lists, tuples, at dictionaries.

Habang gumagawa tayo ng programs, nagde-design tayo ng data structures at sumusulat ng code para manipulahin ang mga data structures na iyon. Mayroong maraming paraan para sumulat ng programs at sa ngayon, malamang nakasulat ka na ng ilang programs na "hindi masyadong elegant" at iba pang programs na "mas elegant". Kahit na ang programs mo ay maaaring maliit, nagsisimula ka nang makita kung paano mayroong kaunting sining at aesthetic sa pagsusulat ng code.

Habang ang programs ay nagiging milyun-milyong linya ang haba, nagiging mas mahalaga na sumulat ng code na madaling maintindihan. Kung nagtatrabaho ka sa million-line program, hindi mo kailanman maaaring panatilihin ang buong program sa isip mo nang sabay. Kailangan natin ng mga paraan para hatiin ang malalaking programs sa maraming mas maliliit na piraso para mas kaunti ang dapat tingnan kapag nagso-solve ng problema, nagfi-fix ng bug, o nagdaragdag ng bagong feature.

Sa isang paraan, ang object oriented programming ay paraan para ayusin ang code mo para maaari mong i-zoom ang 50 lines ng code at maintindihan ito habang hindi pinapansin ang iba pang 999,950 lines ng code sa sandaling iyon.

14.2 Getting started

Tulad ng maraming aspeto ng programming, kailangan na matutunan ang mga konsepto ng object oriented programming bago mo magamit ang mga ito nang epektibo. Dapat mong lapitan ang chapter na ito bilang paraan para matuto ng ilang terms at concepts at magtrabaho sa ilang simpleng halimbawa para maglagay ng pundasyon para sa pag-aaral sa hinaharap.

Ang pangunahing resulta ng chapter na ito ay magkaroon ng basic na pag-unawa kung paano ginagawa ang objects at kung paano sila gumagana at pinakamahalaga kung paano ginagamit natin ang capabilities ng objects na ibinigay sa atin ng Python at Python libraries.

14.3 Using objects

Tulad ng lumalabas, gumagamit na tayo ng objects sa buong libro na ito. Ang Python ay nagbibigay sa atin ng maraming built-in objects. Narito ang ilang simpleng code kung saan ang unang ilang linya ay dapat pakiramdam na napakasimple at natural sa iyo.

```
stuff = list()
stuff.append('python')
stuff.append('chuck')
stuff.sort()
print (stuff[0])
print (stuff.__getitem__(0))
print (list.__getitem__(stuff,0))

# Code: https://www.py4e.com/code3/objects.py
```

Sa halip na tumuon sa kung ano ang nagagawa ng mga linyang ito, tingnan natin kung ano ang talagang nangyayari mula sa punto de vista ng object-oriented programming. Huwag mag-alala kung ang sumusunod na paragraphs ay hindi makakasensya sa unang pagkakataon na binabasa mo sila dahil hindi pa natin tinukoy ang lahat ng terms na ito.

Ang unang linya ay *gumagawa* ng object ng type na `list`, ang pangalawa at pangatlong linya ay *tumatawag* sa `append()` *method*, ang ikaapat na linya ay tumatawag sa `sort()` method, at ang ikalimang linya ay *kumukuha* ng item sa posisyon 0.

Ang ikaanim na linya ay tumatawag sa `__getitem__()` method sa `stuff` list na may parameter na zero.

```
print (stuff.__getitem__(0))
```

Ang ikapitong linya ay mas verbose na paraan ng pagkuha ng 0th item sa list.

```
print (list.__getitem__(stuff,0))
```

Sa code na ito, tumatawag tayo sa `__getitem__` method sa `list` class at *nagpapasa* ng list at item na gusto nating kunin mula sa list bilang parameters.

Ang huling tatlong linya ng program ay katumbas, pero mas maginhawa na simpleng gamitin ang square bracket syntax para maghanap ng item sa partikular na posisyon sa list.

Maaari tayong tumingin sa capabilities ng object sa pamamagitan ng pagtingin sa output ng `dir()` function:

```
>>> stuff = list()
>>> dir(stuff)
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Ang natitirang parte ng chapter na ito ay tutukuyin ang lahat ng terms sa itaas kaya siguraduhing bumalik pagkatapos mong tapusin ang chapter at basahin ulit ang paragraphs sa itaas para suriin ang iyong pag-unawa.

14.4 Starting with programs

Ang program sa pinakabasic na form nito ay tumatanggap ng ilang input, gumagawa ng ilang processing, at gumagawa ng ilang output. Ang elevator conversion program natin ay nagpapakita ng napakaikli pero kumpletong program na nagpapakita ng lahat ng tatlong hakbang na ito.

```
usf = input('Enter the US Floor Number: ')
wf = int(usf) - 1
print('Non-US Floor Number is',wf)

# Code: https://www.py4e.com/code3/elev.py
```

Kung mag-iisip tayo nang kaunti tungkol sa program na ito, mayroong “outside world” at ang program. Ang input at output aspects ay kung saan nakikipagugnayan ang program sa outside world. Sa loob ng program mayroon tayong code at data para makamit ang gawain na idinisenyo ng program na solusyonan.

Ang isang paraan para isipin ang object-oriented programming ay hinihiwalay nito ang program natin sa maraming “zones.” Ang bawat zone ay naglalaman ng ilang code at data (tulad ng program) at may well defined interactions sa outside world at iba pang zones sa loob ng program.

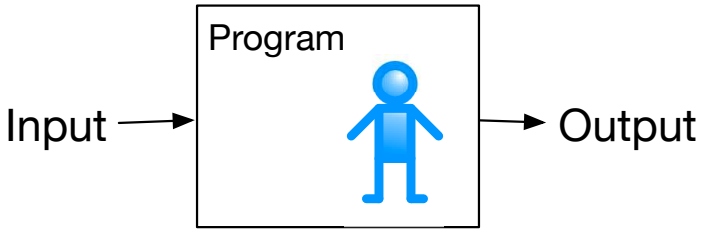

Figure 14.1: A Program

Kung titingnan natin ang link extraction application kung saan ginamit natin ang BeautifulSoup library, makikita natin ang program na ginawa sa pamamagitan ng pagkokonekta ng iba't ibang objects nang magkasama para makamit ang gawain:

```
# To run this, download the BeautifulSoup zip file
# http://www.py4e.com/code3/bs4.zip
# or pip install beautifulsoup4 to ensure you have the latest version
# and unzip it in the same directory as this file

import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl # defauts to certicate verification and most secure protocol (now TLS)

# Ignore SSL/TLS certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))

# Code: https://www.py4e.com/code3/urllinks.py
```

Binabasa natin ang URL sa string at pagkatapos ipinapasa iyon sa `urllib` para kunin ang data mula sa web. Ang `urllib` library ay gumagamit ng `socket` library para gumawa ng aktwal na network connection para kunin ang data. Kinukuha natin ang string na ibinabalik ng `urllib` at ibinibigay sa BeautifulSoup para i-parse. Ang BeautifulSoup ay gumagamit ng object na `html.parser`[1] at nagre-return ng object. Tumatawag tayo sa `tags()` method sa returned object na nagre-return ng dictionary ng tag objects. Naglo-loop tayo sa tags at tumatawag sa `get()` method para sa bawat tag para mag-print ng `href` attribute.

[1] https://docs.python.org/3/library/html.parser.html

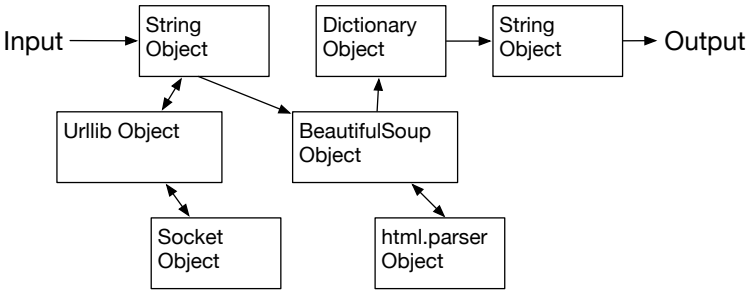

Figure 14.2: A Program as Network of Objects

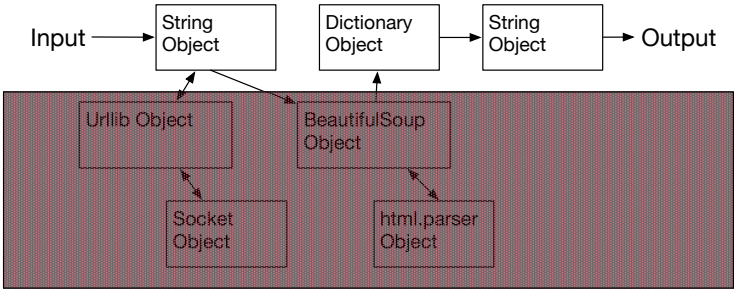

Figure 14.3: Ignoring Detail When Using an Object

Maaari tayong gumuhit ng larawan ng program na ito at kung paano gumagana nang magkasama ang objects.

Ang susi dito ay hindi maintindihan nang perpekto kung paano gumagana ang program na ito pero makita kung paano gumagawa tayo ng network ng interacting objects at nag-o-orchestrate ng paggalaw ng impormasyon sa pagitan ng objects para gumawa ng program. Mahalaga rin na tandaan na kapag tiningnan mo ang program na iyon ilang chapters pabalik, maaari mong ganap na maintindihan kung ano ang nangyayari sa program nang hindi man lang napagtanto na ang program ay "nag-o-orchestrate ng paggalaw ng data sa pagitan ng objects." Ito ay simpleng mga linya ng code na gumagawa ng trabaho.

14.5 Subdividing a problem

Isa sa mga advantages ng object-oriented approach ay maaari nitong itago ang complexity. Halimbawa, habang kailangan nating malaman kung paano gamitin ang `urllib` at BeautifulSoup code, hindi natin kailangang malaman kung paano gumagana ang mga libraries na iyon internally. Nagpapahintulot ito sa atin na tumuon sa parte ng problema na kailangan nating solusyonan at huwag pansinin ang iba pang parte ng program.

Ang kakayahang ito na tumuon eksklusibo sa parte ng program na pinapahalagahan natin at huwag pansinin ang natitira ay kapaki-pakinabang din sa mga developers ng objects na ginagamit natin. Halimbawa, ang mga programmers na gumagawa ng BeautifulSoup ay hindi kailangang malaman o mag-alala tungkol sa

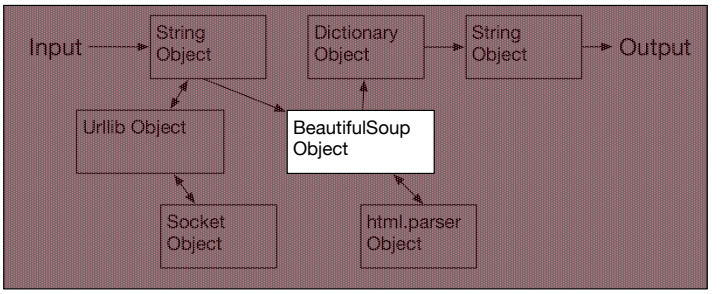

Figure 14.4: Ignoring Detail When Building an Object

kung paano natin kinukuha ang HTML page natin, kung ano ang mga parte na gusto nating basahin, o kung ano ang plano nating gawin sa data na kinukuha natin mula sa web page.

14.6 Our first Python object

Sa basic level, ang object ay simpleng ilang code kasama ang data structures na mas maliit kaysa sa buong program. Ang pagtukoy ng function ay nagpapahintulot sa atin na mag-store ng kaunting code at bigyan ito ng pangalan at pagkatapos tawagin ang code na iyon gamit ang pangalan ng function.

Ang object ay maaaring maglalaman ng ilang functions (na tinatawag nating *methods*) pati na rin ng data na ginagamit ng mga functions na iyon. Tinatawag natin ang data items na parte ng object na *attributes*.

Ginagamit natin ang `class` keyword para tukuyin ang data at code na gagawin ng bawat isa sa objects. Ang class keyword ay kasama ang pangalan ng class at nagsisimula ng indented block ng code kung saan isinasama natin ang attributes (data) at methods (code).

```
class PartyAnimal:

   def __init__(self):
     self.x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()
an.party()
an.party()
an.party()

# Code: https://www.py4e.com/code3/party2.py
```

Ang bawat method ay mukhang function, nagsisimula sa `def` keyword at binubuo ng indented block ng code.

Figure 14.5: A Class and Two Objects

Ang unang method ay specially-named method na tinatawag na `__init__()`. Ang method na ito ay tinatawag para gumawa ng anumang initial setup ng data na gusto nating i-store sa object. Sa class na ito nag-a-allocate tayo ng `x` attribute gamit ang dot notation at ini-initialize ito sa zero.

```
    self.x = 0
```

Ang iba pang method na pinangalanang `party`. Ang lahat ng methods ay may espesyal na unang parameter na pinangalanan natin ayon sa convention na `self`. Ang unang parameter ay nagbibigay sa atin ng access sa object instance para maaari tayong mag-set ng attributes at tumawag ng methods gamit ang dot notation.

Tulad ng `def` keyword ay hindi nagdudulot na ma-execute ang function code, ang `class` keyword ay hindi gumagawa ng object. Sa halip, ang `class` keyword ay nagde-define ng template na nagpapahiwatig kung ano ang data at code na maglalaman sa bawat object ng type na `PartyAnimal`. Ang class ay parang cookie cutter at ang objects na ginawa gamit ang class ay ang cookies[2]. Hindi mo nilalagay ang frosting sa cookie cutter; nilalagay mo ang frosting sa cookies, at maaari kang maglagay ng iba't ibang frosting sa bawat cookie.

Kung magpapatuloy tayo sa sample program na ito, makikita natin ang unang executable line ng code:

```
an = PartyAnimal()
```

Dito natin ini-instruct ang Python na gumawa (i.e., create) ng *object* o *instance* ng class na `PartyAnimal`. Mukha itong function call sa class mismo. Ang Python ay gumagawa ng object na may tamang data at methods at nagre-return ng object na pagkatapos ay na-a-assign sa variable na `an`. Sa isang paraan ito ay medyo katulad ng sumusunod na linya na ginagamit natin sa buong panahon:

```
counts = dict()
```

[2]Cookie image copyright CC-BY https://www.flickr.com/photos/dinnerseries/23570475099

Dito ini-instruct natin ang Python na gumawa ng object gamit ang `dict` template (nandito na sa Python), ibalik ang instance ng dictionary, at i-assign ito sa variable na `counts`.

Kapag ang `PartyAnimal` class ay ginagamit para gumawa ng object, ang variable na `an` ay ginagamit para tumuro sa object na iyon. Ginagamit natin ang `an` para ma-access ang code at data para sa partikular na instance ng `PartyAnimal` class.

Ang bawat Partyanimal object/instance ay naglalaman sa loob nito ng variable na `x` at method/function na pinangalanang `party`. Tumatawag tayo sa `party` method sa linyang ito:

```
an.party()
```

Kapag ang `party` method ay tinatawag, ang unang parameter (na tinatawag natin ayon sa convention na `self`) ay tumuturo sa partikular na instance ng PartyAnimal object na tinatawag ang `party`. Sa loob ng `party` method, nakikita natin ang linya:

```
self.x = self.x + 1
```

Ang syntax na ito gamit ang *dot* operator ay nagsasabing 'ang x sa loob ng self.' Sa bawat pagkakataon na tinatawag ang `party()`, ang internal `x` value ay na-i-increment ng 1 at ang value ay na-pi-print.

Ang sumusunod na linya ay isa pang paraan para tawagin ang `party` method sa loob ng `an` object:

```
PartyAnimal.party(an)
```

Sa variation na ito, na-a-access natin ang code mula sa loob ng class at tahasang ipinapasa ang object pointer na `an` bilang unang parameter (i.e., `self` sa loob ng method). Maaari mong isipin ang `an.party()` bilang shorthand para sa linya sa itaas.

Kapag nag-e-execute ang program, gumagawa ito ng sumusunod na output:

```
So far 1
So far 2
So far 3
So far 4
```

Ang object ay ginawa, at ang `party` method ay tinawag nang apat na beses, pare-hong nag-i-increment at nagpi-print ng value para sa `x` sa loob ng `an` object.

14.7 Classes as types

Tulad ng nakita natin, sa Python lahat ng variables ay may type. Maaari nating gamitin ang built-in `dir` function para suriin ang capabilities ng variable. Maaari din nating gamitin ang `type` at `dir` sa mga classes na ginagawa natin.

```
class PartyAnimal:

   def __init__(self):
     self.x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()
print ("Type", type(an))
print ("Dir ", dir(an))
print ("Type", type(an.x))
print ("Type", type(an.party))

# Code: https://www.py4e.com/code3/party3.py
```

Kapag nag-e-execute ang program na ito, gumagawa ito ng sumusunod na output:

```
Type <class '__main__.PartyAnimal'>
Dir  ['__class__', '__delattr__', ...
'__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'party', 'x']
Type <class 'int'>
Type <class 'method'>
```

Makikita mo na gamit ang `class` keyword, gumawa tayo ng bagong type. Mula sa `dir` output, makikita mo na parehong `x` integer attribute at `party` method ay available sa object.

14.8 Object lifecycle

Sa naunang mga halimbawa, nagde-define tayo ng class (template), gumagamit ng class na iyon para gumawa ng instance ng class na iyon (object), at pagkatapos gumagamit ng instance. Kapag natapos ang program, lahat ng variables ay itinatapon. Karaniwan, hindi natin masyadong iniisip ang paggawa at pagkasira ng variables, pero kadalasan habang ang objects natin ay nagiging mas kumplikado, kailangan nating gumawa ng ilang aksyon sa loob ng object para mag-set up ng mga bagay habang ginagawa ang object at posibleng linisin ang mga bagay kapag ang object ay itinapon.

Kung gusto nating malaman ng object natin ang mga sandaling ito ng construction at destruction, nagdaragdag tayo ng specially named methods sa object natin:

```
class PartyAnimal:

   def __init__(self):
     self.x = 0
     print('I am constructed')
```

```
    def party(self) :
      self.x = self.x + 1
      print('So far',self.x)

    def __del__(self):
      print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)

# Code: https://www.py4e.com/code3/party4.py
```

Kapag nag-e-execute ang program na ito, gumagawa ito ng sumusunod na output:

```
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

Habang gumagawa ang Python ng object natin, tinatawag nito ang `__init__` method natin para bigyan tayo ng pagkakataon na mag-set up ng ilang default o initial values para sa object. Kapag nakakita ang Python ng linya:

```
an = 42
```

Talagang "itinatapon ang object natin" para maaari nitong muling gamitin ang variable na `an` para mag-store ng value na `42`. Sa sandaling ang `an` object natin ay "sinisira" ang destructor code natin (`__del__`) ay tinatawag. Hindi natin mapipigilan ang variable natin na masira, pero maaari tayong gumawa ng anumang kailangan cleanup bago lang mawala ang object natin.

Kapag gumagawa ng objects, napakakaraniwan na magdagdag ng constructor sa object para mag-set up ng initial values para sa object. Relatively bihira na kailangan ng destructor para sa object.

14.9 Multiple instances

Hanggang ngayon, nagde-define tayo ng class, gumawa ng isang object, gumamit ng object na iyon, at pagkatapos itinapon ang object. Gayunpaman, ang tunay na kapangyarihan sa object-oriented programming ay nangyayari kapag gumagawa tayo ng maraming instances ng class natin.

Kapag gumagawa tayo ng maraming objects mula sa class natin, maaaring gusto nating mag-set up ng iba't ibang initial values para sa bawat isa sa objects. Maaari tayong magpasa ng data sa constructors para bigyan ang bawat object ng iba't ibang initial value:

```
class PartyAnimal:

   def __init__(self, nam):
     self.x = 0
     self.name = nam
     print(self.name,'constructed')

   def party(self) :
     self.x = self.x + 1
     print(self.name,'party count',self.x)

s = PartyAnimal('Sally')
s.party()
j = PartyAnimal('Jim')

j.party()
s.party()
```

 # Code: https://www.py4e.com/code3/party5.py

Ang constructor ay may parehong `self` parameter na tumuturo sa object instance at karagdagang parameters na ipinapasa sa constructor habang ginagawa ang object:

```
s = PartyAnimal('Sally')
```

Sa loob ng constructor, ang pangalawang linya ay kumokopya ng parameter (`nam`) na ipinapasa sa `name` attribute sa loob ng object instance.

```
self.name = nam
```

Ang output ng program ay nagpapakita na ang bawat isa sa objects (`s` at `j`) ay naglalaman ng kanilang sariling independent copies ng `x` at `nam`:

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Sally party count 2
```

14.10 Inheritance

Ang isa pang makapangyarihang feature ng object-oriented programming ay ang kakayahang gumawa ng bagong class sa pamamagitan ng pag-extend ng existing class. Kapag nag-e-extend ng class, tinatawag natin ang original class na *parent class* at ang bagong class na *child class*.

Para sa halimbawang ito, inililipat natin ang `PartyAnimal` class natin sa sarili nitong file. Pagkatapos, maaari nating 'i-import' ang `PartyAnimal` class sa bagong file at i-extend ito, tulad ng sumusunod:

```
from party import PartyAnimal

class CricketFan(PartyAnimal):

   def __init__(self, nam) :
       super().__init__(nam)
       self.points = 0

   def six(self):
      self.points = self.points + 6
      self.party()
      print(self.name,"points",self.points)

s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
print(dir(j))

# Code: https://www.py4e.com/code3/party6.py
```

Kapag nagde-define tayo ng `CricketFan` class, ipinapahiwatig natin na nag-e-extend tayo sa `PartyAnimal` class. Nangangahulugan ito na lahat ng variables (`x`) at methods (`party`) mula sa `PartyAnimal` class ay *na-inherit* ng `CricketFan` class. Halimbawa, sa loob ng `six` method sa `CricketFan` class, tumatawag tayo sa `party` method mula sa `PartyAnimal` class.

Gumagamit tayo ng espesyal na syntax sa `__init__()` method sa `CricketFan` class para siguraduhin na tinatawag natin ang `__init__()` method sa `PartyAnimal` para ang anumang setup na kailangan ng `PartyAnimal` ay ginagawa bilang karagdagan sa setup na kailangan para sa `CricketFan` extensions.

```
   def __init__(self, nam) :
       super().__init__(nam)
       self.points = 0
```

Ang `super()` syntax ay nagsasabi sa Python na tawagin ang `__init__` method sa class na ine-extend natin. Ang `PartyAnimal` ay ang super (o parent) class at ang `CricketFan` ay ang sub (o child) class.

Habang nag-e-execute ang program, gumagawa tayo ng `s` at `j` bilang independent instances ng `PartyAnimal` at `CricketFan`. Ang `j` object ay may karagdagang capabilities lampas sa `s` object.

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
['__class__', '__delattr__', ... '__weakref__',
'name', 'party', 'points', 'six', 'x']
```

Sa `dir` output para sa `j` object (instance ng `CricketFan` class), nakikita natin na mayroon itong attributes at methods ng parent class, pati na rin ang attributes at methods na idinagdag kapag ang class ay na-extend para gumawa ng `CricketFan` class.

14.11 Summary

Ito ay napakabilis na pagpapakilala sa object-oriented programming na nakatuon mainly sa terminology at syntax ng pagtukoy at paggamit ng objects. Mabilis nating suriin ang code na tiningnan natin sa simula ng chapter. Sa puntong ito dapat mong ganap na maintindihan kung ano ang nangyayari.

```
stuff = list()
stuff.append('python')
stuff.append('chuck')
stuff.sort()
print (stuff[0])
print (stuff.__getitem__(0))
print (list.__getitem__(stuff,0))

# Code: https://www.py4e.com/code3/objects.py
```

Ang unang linya ay gumagawa ng `list` *object*. Kapag gumagawa ang Python ng `list` object, tinatawag nito ang *constructor* method (pinangalanang `__init__`) para mag-set up ng internal data attributes na gagamitin para mag-store ng list data. Hindi tayo nagpasa ng anumang parameters sa *constructor*. Kapag nagre-return ang constructor, ginagamit natin ang variable na `stuff` para tumuro sa returned instance ng `list` class.

Ang pangalawa at pangatlong linya ay tumatawag sa `append` method na may isang parameter para magdagdag ng bagong item sa dulo ng list sa pamamagitan ng pag-update ng attributes sa loob ng `stuff`. Pagkatapos sa ikaapat na linya, tumatawag tayo sa `sort` method na walang parameters para i-sort ang data sa loob ng `stuff` object.

Pagkatapos nagpi-print tayo ng unang item sa list gamit ang square brackets na shortcut para tawagin ang `__getitem__` method sa loob ng `stuff`. Ito ay katumbas ng pagtawag sa `__getitem__` method sa `list` *class* at pagpasa ng `stuff` object bilang unang parameter at ang posisyon na hinahanap natin bilang pangalawang parameter.

Sa dulo ng program, ang `stuff` object ay itinatapon pero hindi bago tawagin ang *destructor* (pinangalanang `__del__`) para ang object ay makapaglinis ng anumang loose ends kung kinakailangan.

Iyon ang basics ng object-oriented programming. Mayroong maraming karagdagang detalye tungkol sa kung paano pinakamahusay na gamitin ang object-oriented approaches kapag gumagawa ng malalaking applications at libraries na lampas sa saklaw ng chapter na ito.[3]

[3]Kung curious ka tungkol sa kung saan tinukoy ang `list` class, tingnan (inaasahan na hindi

14.12 Glossary

attribute Variable na parte ng class.

class Template na maaaring gamitin para gumawa ng object. Nagde-define ng attributes at methods na gagawin ng object.

child class Bagong class na ginawa kapag ang parent class ay na-extend. Ang child class ay nag-i-inherit ng lahat ng attributes at methods ng parent class.

constructor Opsiyonal na specially named method (`__init__`) na tinatawag sa sandaling ang class ay ginagamit para gumawa ng object. Karaniwang ginagamit ito para mag-set up ng initial values para sa object.

destructor Opsiyonal na specially named method (`__del__`) na tinatawag sa sandaling bago masira ang object. Ang destructors ay bihirang ginagamit.

inheritance Kapag gumagawa tayo ng bagong class (child) sa pamamagitan ng pag-extend ng existing class (parent). Ang child class ay may lahat ng attributes at methods ng parent class kasama ang karagdagang attributes at methods na tinukoy ng child class.

method Function na nakapaloob sa class at objects na ginawa mula sa class. Ang ilang object-oriented patterns ay gumagamit ng 'message' sa halip na 'method' para ilarawan ang konseptong ito.

object Constructed instance ng class. Ang object ay naglalaman ng lahat ng attributes at methods na tinukoy ng class. Ang ilang object-oriented documentation ay gumagamit ng term na 'instance' nang magkakapalit sa 'object'.

parent class Ang class na ine-extend para gumawa ng bagong child class. Ang parent class ay nag-a-ambag ng lahat ng methods at attributes nito sa bagong child class.

---

magbabago ang URL) https://github.com/python/cpython/blob/master/Objects/listobject.c - ang list class ay isinulat sa language na tinatawag na "C". Kung titingnan mo ang source code na iyon at makita mo itong curious maaaring gusto mong i-explore ang C programming na may pagtingin sa paggawa ng objects sa https://www.cc4e.com/.

Chapter 15

Using Databases and SQL

15.1 What is a database?

Ang *database* ay file na inoorganisa para mag-store ng data. Karamihan sa databases ay inoorganisa tulad ng dictionary sa diwa na nagma-map sila mula sa keys patungo sa values. Ang pinakamalaking pagkakaiba ay ang database ay nasa disk (o iba pang permanent storage), kaya ito ay nananatili pagkatapos matapos ang program. Dahil ang database ay naka-store sa permanent storage, maaari itong mag-store ng mas maraming data kaysa sa dictionary, na limitado sa laki ng memory sa computer.

Tulad ng dictionary, ang database software ay idinisenyo para panatilihing napakabilis ang pag-insert at pag-access ng data, kahit para sa malalaking dami ng data. Ang database software ay nagma-maintain ng performance nito sa pamamagitan ng paggawa ng *indexes* habang idinadagdag ang data sa database para payagan ang computer na tumalon nang mabilis sa partikular na entry.

Mayroong maraming iba't ibang database systems na ginagamit para sa malawak na iba't ibang layunin kasama ang: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, at SQLite. Nakatuon tayo sa SQLite sa libro na ito dahil ito ay napakakaraniwang database at naka-build na sa Python. Ang SQLite ay idinisenyo para ma-*embed* sa iba pang applications para magbigay ng database support sa loob ng application. Halimbawa, ang Firefox browser ay gumagamit din ng SQLite database internally tulad ng maraming iba pang produkto.

http://sqlite.org/

Ang SQLite ay angkop sa ilan sa data manipulation problems na nakikita natin sa Informatics.

15.2 Database concepts

Kapag unang tiningnan mo ang database, mukha itong spreadsheet na may maraming sheets. Ang pangunahing data structures sa database ay: *tables*, *rows*, at *columns*.

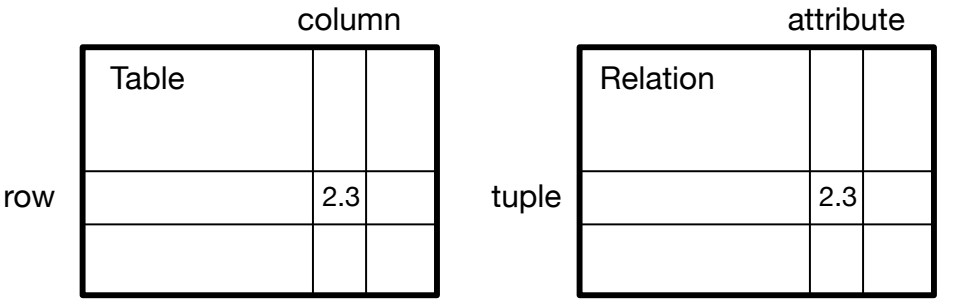

Figure 15.1: Relational Databases

Sa teknikal na mga paglalarawan ng relational databases ang mga konsepto ng table, row, at column ay mas pormal na tinutukoy bilang *relation*, *tuple*, at *attribute*, ayon sa pagkakabanggit. Gagamitin natin ang mas hindi pormal na terms sa chapter na ito.

15.3 Database Browser for SQLite

Habang ang chapter na ito ay nakatuon sa paggamit ng Python para magtrabaho sa data sa SQLite database files, maraming operations ay maaaring gawin nang mas maginhawa gamit ang software na tinatawag na *Database Browser for SQLite* na libreng available mula sa:

http://sqlitebrowser.org/

Gamit ang browser maaari mong madaling gumawa ng tables, mag-insert ng data, mag-edit ng data, o magpatakbo ng simpleng SQL queries sa data sa database.

Sa isang diwa, ang database browser ay katulad ng text editor kapag nagtatrabaho sa text files. Kapag gusto mong gumawa ng isa o napakakaunting operations sa text file, maaari mo lang itong buksan sa text editor at gawin ang mga pagbabagong gusto mo. Kapag mayroon kang maraming pagbabago na kailangan mong gawin sa text file, kadalasan susulat ka ng simpleng Python program. Makikita mo ang parehong pattern kapag nagtatrabaho sa databases. Gagawa ka ng simpleng operations sa database manager at mas kumplikadong operations ay pinaka-maginhawang gawin sa Python.

15.4 Creating a database table

Ang databases ay nangangailangan ng mas tinukoy na structure kaysa sa Python lists o dictionaries[1].

Kapag gumagawa tayo ng database *table* dapat nating sabihin sa database nang maaga ang mga pangalan ng bawat isa sa *columns* sa table at ang uri ng data na

[1]Ang SQLite ay talagang nagpapahintulot ng ilang flexibility sa uri ng data na naka-store sa column, pero pananatilihin natin ang data types natin na strict sa chapter na ito para ang concepts ay nalalapat nang pantay sa iba pang database systems tulad ng MySQL.

plano nating i-store sa bawat *column*. Kapag alam ng database software ang uri ng data sa bawat column, maaari nitong piliin ang pinakamabisang paraan para mag-store at maghanap ng data batay sa uri ng data.

Maaari mong tingnan ang iba't ibang data types na sinusuportahan ng SQLite sa sumusunod na url:

http://www.sqlite.org/datatypes.html

Ang pagtukoy ng structure para sa data mo nang maaga ay maaaring mukhang hindi maginhawa sa simula, pero ang kapalit ay mabilis na access sa data mo kahit na ang database ay naglalaman ng malaking dami ng data.

Ang code para gumawa ng database file at table na pinangalanang `Track` na may dalawang columns sa database ay ganito:

```
import sqlite3

conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Track')
cur.execute('CREATE TABLE Track (title TEXT, plays INTEGER)')

conn.close()

# Code: https://www.py4e.com/code3/db1.py
```

Ang `connect` operation ay gumagawa ng "connection" sa database na naka-store sa file na `music.sqlite` sa current directory. Kung ang file ay hindi umiiral, gagawin ito. Ang dahilan kung bakit ito ay tinatawag na "connection" ay dahil minsan ang database ay naka-store sa hiwalay na "database server" mula sa server kung saan tumatakbo ang aming application. Sa simpleng halimbawa natin ang database ay simpleng local file sa parehong directory ng Python code na tumatakbo tayo.

Ang *cursor* ay parang file handle na maaari nating gamitin para gumawa ng operations sa data na naka-store sa database. Ang pagtawag sa `cursor()` ay napakatulad conceptually sa pagtawag sa `open()` kapag nakikipag-ugnayan sa text files.

Kapag mayroon na tayong cursor, maaari na tayong magsimulang mag-execute ng commands sa contents ng database gamit ang `execute()` method.

Ang database commands ay ipinahayag sa espesyal na language na na-standardize sa maraming iba't ibang database vendors para payagan tayong matuto ng isang database language. Ang database language ay tinatawag na *Structured Query Language* o *SQL* para sa maikli.

http://en.wikipedia.org/wiki/SQL

Sa halimbawa natin, nag-e-execute tayo ng dalawang SQL commands sa database natin. Bilang convention, ipapakita natin ang SQL keywords sa uppercase at ang mga parte ng command na idinagdag natin (tulad ng table at column names) ay ipapakita sa lowercase.

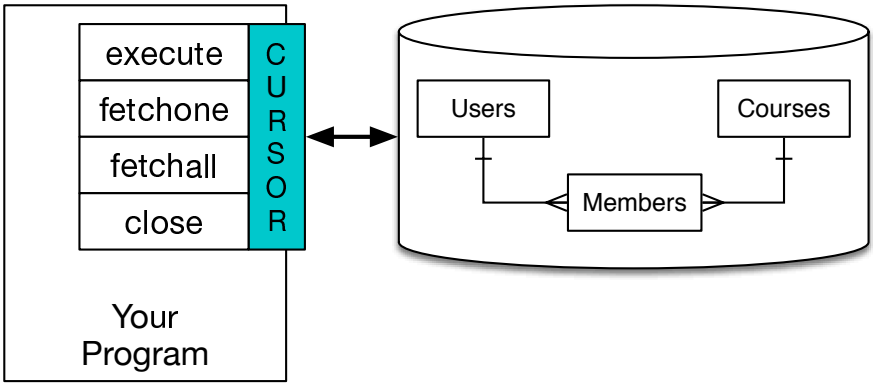

Figure 15.2: A Database Cursor

Ang unang SQL command ay nagtatanggal ng `Track` table mula sa database kung umiiral ito. Ang pattern na ito ay simpleng para payagan tayong patakbuhin ang parehong program para gumawa ng `Track` table nang paulit-ulit nang hindi nagdudulot ng error. Tandaan na ang `DROP TABLE` command ay nagtatanggal ng table at lahat ng contents nito mula sa database (i.e., walang "undo").

```
cur.execute('DROP TABLE IF EXISTS Track ')
```

Ang pangalawang command ay gumagawa ng table na pinangalanang `Track` na may text column na pinangalanang `title` at integer column na pinangalanang `plays`.

```
cur.execute('CREATE TABLE Track (title TEXT, plays INTEGER)')
```

Ngayon na gumawa na tayo ng table na pinangalanang `Track`, maaari na tayong maglagay ng ilang data sa table na iyon gamit ang SQL `INSERT` operation. Muli, nagsisimula tayo sa pamamagitan ng paggawa ng connection sa database at pagkuha ng `cursor`. Pagkatapos maaari na tayong mag-execute ng SQL commands gamit ang cursor.

Ang SQL `INSERT` command ay nagpapahiwatig kung aling table ang ginagamit natin at pagkatapos nagde-define ng bagong row sa pamamagitan ng pagli-list ng fields na gusto nating isama `(title, plays)` na sinusundan ng `VALUES` na gusto nating ilagay sa bagong row. Tinutukoy natin ang values bilang question marks `(?, ?)` para ipahiwatig na ang aktwal na values ay ipinapasa bilang tuple `( 'My Way', 15 )` bilang pangalawang parameter sa `execute()` call.

```
import sqlite3

conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()

cur.execute('INSERT INTO Track (title, plays) VALUES (?, ?)',
    ('Thunderstruck', 20))
```

Tracks

title	plays
Thunderstruck	20
My Way	15

Figure 15.3: Rows in a Table

```
cur.execute('INSERT INTO Track (title, plays) VALUES (?, ?)',
    ('My Way', 15))
conn.commit()

print('Track:')
cur.execute('SELECT title, plays FROM Track')
for row in cur:
     print(row)

cur.execute('DELETE FROM Track WHERE plays < 100')
conn.commit()

cur.close()

# Code: https://www.py4e.com/code3/db2.py
```

Una nag-`INSERT` tayo ng dalawang rows sa table natin at gumagamit ng `commit()` para pilitin ang data na isulat sa database file.

Pagkatapos gumagamit tayo ng `SELECT` command para kunin ang rows na kakalagay lang natin mula sa table. Sa `SELECT` command, ipinapahiwatig natin kung aling columns ang gusto natin `(title, plays)` at ipinapahiwatig kung aling table ang gusto nating kunin ang data. Pagkatapos mag-execute ng `SELECT` statement, ang cursor ay isang bagay na maaari nating loop through sa `for` statement. Para sa efficiency, ang cursor ay hindi nagbabasa ng lahat ng data mula sa database kapag nag-e-execute tayo ng `SELECT` statement. Sa halip, ang data ay binabasa on demand habang naglo-loop tayo sa rows sa `for` statement.

Ang output ng program ay ganito:

```
Track:
('Thunderstruck', 20)
('My Way', 15)
```

Ang `for` loop natin ay nakakahanap ng dalawang rows, at ang bawat row ay Python tuple na may unang value bilang `title` at pangalawang value bilang bilang ng `plays`.

Sa pinakadulo ng program, nag-e-execute tayo ng SQL command para `DELETE` ang rows na kakalikha lang natin para maaari nating patakbuhin ang program

nang paulit-ulit. Ang `DELETE` command ay nagpapakita ng paggamit ng `WHERE` clause na nagpapahintulot sa atin na ipahayag ang selection criterion para maaari nating hilingin sa database na ilapat ang command lang sa rows na tumutugma sa criterion. Sa halimbawang ito ang criterion ay nalalapat sa lahat ng rows kaya inu-emptying natin ang table para maaari nating patakbuhin ang program nang paulit-ulit. Pagkatapos gawin ang `DELETE`, tumatawag din tayo sa `commit()` para pilitin ang data na matanggal mula sa database.

15.5 Structured Query Language summary

Hanggang ngayon, gumagamit tayo ng Structured Query Language sa aming Python examples at natakpan na natin ang marami sa basics ng SQL commands. Sa section na ito, titingnan natin ang SQL language partikular at magbigay ng overview ng SQL syntax.

Dahil mayroong napakaraming iba't ibang database vendors, ang Structured Query Language (SQL) ay na-standardize para maaari tayong makipag-communicate sa portable na paraan sa database systems mula sa maraming vendors.

Ang relational database ay binubuo ng tables, rows, at columns. Ang columns ay karaniwang may type tulad ng text, numeric, o date data. Kapag gumagawa tayo ng table, ipinapahiwatig natin ang mga pangalan at uri ng columns:

```
CREATE TABLE Track (title TEXT, plays INTEGER)
```

Para mag-insert ng row sa table, gumagamit tayo ng SQL `INSERT` command:

```
INSERT INTO Track (title, plays) VALUES ('My Way', 15)
```

Ang `INSERT` statement ay nagtutukoy ng table name, pagkatapos list ng fields/columns na gusto mong i-set sa bagong row, at pagkatapos ang keyword na `VALUES` at list ng katumbas na values para sa bawat field.

Ang SQL `SELECT` command ay ginagamit para kunin ang rows at columns mula sa database. Ang `SELECT` statement ay nagpapahintulot sa iyo na tukuyin kung aling columns ang gusto mong kunin pati na rin ang `WHERE` clause para piliin kung aling rows ang gusto mong makita. Nagpapahintulot din ito ng opsiyonal na `ORDER BY` clause para kontrolin ang sorting ng returned rows.

```
SELECT * FROM Track WHERE title = 'My Way'
```

Ang paggamit ng `*` ay nagpapahiwatig na gusto mong ibalik ng database ang lahat ng columns para sa bawat row na tumutugma sa `WHERE` clause.

Tandaan, hindi tulad sa Python, sa SQL `WHERE` clause gumagamit tayo ng isang equal sign para ipahiwatig ang test para sa equality sa halip na double equal sign. Ang iba pang logical operations na pinapayagan sa `WHERE` clause ay kasama ang `<`, `>`, `<=`, `>=`, `!=`, pati na rin ang `AND` at `OR` at parentheses para gumawa ng logical expressions mo.

Maaari mong hilingin na ang returned rows ay ma-sort ayon sa isa sa fields tulad ng sumusunod:

```
SELECT title,plays FROM Track ORDER BY title
```

Posibleng UPDATE ang column o columns sa loob ng isa o higit pang rows sa table gamit ang SQL UPDATE statement tulad ng sumusunod:

```
UPDATE Track SET plays = 16 WHERE title = 'My Way'
```

Ang UPDATE statement ay nagtutukoy ng table at pagkatapos list ng fields at values na baguhin pagkatapos ng SET keyword at pagkatapos opsiyonal na WHERE clause para piliin ang rows na dapat i-update. Ang isang UPDATE statement ay magbabago ng lahat ng rows na tumutugma sa WHERE clause. Kung ang WHERE clause ay hindi tinukoy, ginagawa nito ang UPDATE sa lahat ng rows sa table.

Para tanggalin ang row, kailangan mo ng WHERE clause sa SQL DELETE statement. Ang WHERE clause ay nagde-determine kung aling rows ang dapat tanggalin:

```
DELETE FROM Track WHERE title = 'My Way'
```

Ang apat na basic SQL commands na ito (INSERT, SELECT, UPDATE, at DELETE) ay nagpapahintulot sa apat na basic operations na kailangan para gumawa at mag-maintain ng data. Gumagamit tayo ng "CRUD" (Create, Read, Update, at Delete) para makuha ang lahat ng concepts na ito sa isang term.[2]

15.6 Multiple tables and basic data modeling

Ang tunay na kapangyarihan ng relational database ay kapag gumagawa tayo ng maraming tables at gumagawa ng links sa pagitan ng mga tables na iyon. Ang gawain ng pagde-decide kung paano hatiin ang application data mo sa maraming tables at pagtatatag ng relationships sa pagitan ng tables ay tinatawag na *data modeling*. Ang design document na nagpapakita ng tables at kanilang relationships ay tinatawag na *data model*.

Ang data modeling ay relatively sophisticated skill at ipakikilala lang natin ang pinakabasic concepts ng relational data modeling sa section na ito. Para sa mas detalyado tungkol sa data modeling maaari kang magsimula sa:

http://en.wikipedia.org/wiki/Relational_model

Sabihin natin para sa tracks database natin gusto nating i-track ang pangalan ng `artist` para sa bawat track bilang karagdagan sa `title` at bilang ng plays para sa bawat track. Ang simpleng approach ay maaaring simpleng magdagdag ng iba pang column sa database na tinatawag na `artist` at ilagay ang pangalan ng artist sa column tulad ng sumusunod:

```
DROP TABLE IF EXISTS Track;
CREATE TABLE Track (title TEXT, plays INTEGER, artist TEXT);
```

[2] Oo may disconnect sa pagitan ng "CRUD" term at unang letters ng apat na SQL statements na nag-i-implement ng "CRUD". Ang posibleng paliwanag ay maaaring sabihin na ang "CRUD" ay ang "concept" at ang SQL ay ang implementation. Ang isa pang posibleng paliwanag ay mas masaya sabihin ang "CRUD" kaysa sa "ISUD".

Pagkatapos maaari tayong mag-insert ng ilang tracks sa table natin.

```
INSERT INTO Track (title, plays, artist)
    VALUES ('My Way', 15, 'Frank Sinatra');
INSERT INTO Track (title, plays, artist)
    VALUES ('New York', 25, 'Frank Sinatra');
```

Kung titingnan natin ang data natin gamit ang `SELECT * FROM Track` statement, mukhang maganda ang ginawa natin.

```
sqlite> SELECT * FROM Track;
My Way|15|Frank Sinatra
New York|25|Frank Sinatra
sqlite>
```

Gumawa tayo ng *napakasamang error* sa data modeling natin. Nilabag natin ang rules ng *database normalization*.

https://en.wikipedia.org/wiki/Database_normalization

Habang ang database normalization ay mukhang napakakumplikado sa ibabaw at naglalaman ng maraming mathematical justifications, sa ngayon maaari nating bawasan ang lahat sa isang simpleng rule na susundin natin.

Hindi dapat nating ilagay ang parehong string data sa column nang higit sa isang beses. Kung kailangan natin ang data nang higit sa isang beses, gumagawa tayo ng numeric *key* para sa data at nagre-reference sa aktwal na data gamit ang key na ito. Lalo na kung ang maraming entries ay tumutukoy sa parehong object.

Para ipakita ang slippery slope na binababa natin sa pamamagitan ng pag-assign ng string columns sa database model natin, isipin kung paano natin babaguhin ang data model kung gusto nating i-track ang eye color ng artists natin? Gagawin ba natin ito?

```
DROP TABLE IF EXISTS Track;
CREATE TABLE Track (title TEXT, plays INTEGER,
    artist TEXT, eyes TEXT);
INSERT INTO Track (title, plays, artist, eyes)
    VALUES ('My Way', 15, 'Frank Sinatra', 'Blue');
INSERT INTO Track (title, plays, artist, eyes)
    VALUES ('New York', 25, 'Frank Sinatra', 'Blue');
```

Dahil nag-record si Frank Sinatra ng higit sa 1200 songs, talaga bang ilalagay natin ang string na 'Blue' sa 1200 rows sa `Track` table natin. At ano ang mangyayari kung magde-decide tayo na ang eye color niya ay 'Light Blue'? May bagay na hindi tama.

Ang tamang solusyon ay gumawa ng table para sa bawat `Artist` at mag-store ng lahat ng data tungkol sa artist sa table na iyon. At pagkatapos kailangan nating gumawa ng connection sa pagitan ng row sa `Track` table patungo sa row sa `Artist` table. Marahil maaari nating tawagin ang "link" na ito sa pagitan ng dalawang "tables" na "relationship" sa pagitan ng dalawang tables. At iyon mismo ang napagpasyahan ng database experts na tawagin sa lahat ng mga links na ito.

Gumawa tayo ng `Artist` table tulad ng sumusunod:

```
DROP TABLE IF EXISTS Artist;
CREATE TABLE Artist (name TEXT, eyes TEXT);
INSERT INTO Artist (name, eyes)
   VALUES ('Frank Sinatra', 'blue');
```

Ngayon mayroon tayong dalawang tables pero kailangan natin ng paraan para *i-link* ang rows sa dalawang tables. Para gawin ito, kailangan natin ng tinatawag nating 'keys'. Ang mga keys na ito ay simpleng integer numbers na maaari nating gamitin para maghanap ng row sa iba't ibang table. Kung gagawa tayo ng links sa rows sa loob ng table, kailangan nating magdagdag ng *primary key* sa rows sa table. Ayon sa convention karaniwang pinapangalanan natin ang primary key column na 'id'. Kaya ang `Artist` table natin ay ganito:

```
DROP TABLE IF EXISTS Artist;
CREATE TABLE Artist (id INTEGER, name TEXT, eyes TEXT);
INSERT INTO Artist (id, name, eyes)
   VALUES (42, 'Frank Sinatra', 'blue');
```

Ngayon mayroon tayong row sa table para sa 'Frank Sinatra' (at eye color niya) at primary key na '42' para gamitin para i-link ang tracks natin sa kanya. Kaya binabago natin ang Track table natin tulad ng sumusunod:

```
DROP TABLE IF EXISTS Track;
CREATE TABLE Track (title TEXT, plays INTEGER,
    artist_id INTEGER);
INSERT INTO Track (title, plays, artist_id)
    VALUES ('My Way', 15, 42);
INSERT INTO Track (title, plays, artist_id)
    VALUES ('New York', 25, 42);
```

Ang `artist_id` column ay integer, at ayon sa naming convention ay *foreign key* na tumuturo sa *primary* key sa `Artist` table. Tinatawag natin itong foreign key dahil tumuturo ito sa row sa iba't ibang table.

Ngayon sumusunod na tayo sa rules ng database normalization, pero kapag gusto nating makuha ang data mula sa database natin, hindi natin gusto na makita ang 42, gusto nating makita ang pangalan at eye color ng artist. Para gawin ito gumagamit tayo ng keyword na `JOIN` sa SELECT statement natin.

```
SELECT title, plays, name, eyes
FROM Track JOIN Artist
ON Track.artist_id = Artist.id;
```

Ang `JOIN` clause ay kasama ang `ON` condition na nagde-define kung paano ang rows ay dapat konektado. Para sa bawat row sa `Track` idagdag ang data mula sa `Artist` mula sa row kung saan ang `artist_id` sa `Track` table ay tumutugma sa `id` mula sa `Artist` table.

Ang output ay magiging:

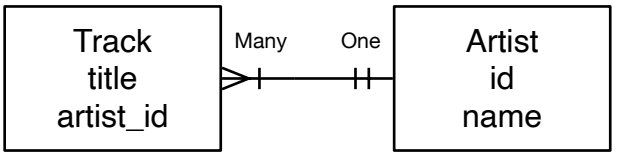

Figure 15.4: A Verbose One-to-Many Data Model

```
My Way|15|Frank Sinatra|blue
New York|25|Frank Sinatra|blue
```

Habang maaaring mukhang medyo clunky at ang instincts mo ay maaaring sabihin sa iyo na mas mabilis lang na panatilihin ang data sa isang table, lumalabas na ang limit sa database performance ay kung gaano karaming data ang kailangang i-scan kapag kumukuha ng query. Habang ang detalye ay napakakumplikado, ang integers ay mas maliit kaysa sa strings (lalo na ang Unicode) at mas mabilis na ilipat at i-compare.

15.7 Data model diagrams

Habang ang `Track` at `Artist` database design natin ay simple na may dalawang tables lang at isang one-to-many relationship, ang mga data models na ito ay maaaring maging kumplikado nang mabilis at mas madaling maintindihan kung maaari tayong gumawa ng graphical representation ng data model natin.

While there are many graphical representations of data models, we will use one of the "classic" approaches, called "Crow's Foot Diagrams" as shown in Figure 15.4. Each table is shown as a box with the name of the table and its columns. Then where there is a relationship between two tables a line is drawn connecting the tables with a notation added to the end of each line indicating the nature of the relationship.

https://en.wikipedia.org/wiki/Entity-relationship_model

In this case, "many" tracks can be associated with each artist. So the track end is shown with the crow's foot spread out indicating it is the" "many" end. The artist end is shown with a vertical like that indicates "one". There will be "many" artists in general, but the important aspect is that for each artist there will be many tracks. And each of those artists may be associated with multiple tracks.

You will note that the column that holds the *foreign_key* like `artist_id` is on the "many" end and the *primary key* is at the "one" end.

Since the pattern of foreign and primary key placement is so consistent and follows the "many" and "one" ends of the lines, we never include either the primary or foreign key columns in our diagram of the data model as shown in the second diagram as shown in Figure 15.5. The columns are thought of as "implementation detail" to capture the nature of the relationship details and not an essential part of the data being modeled.

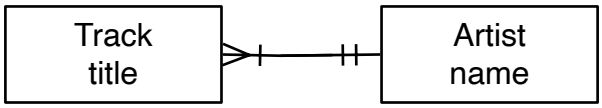

Figure 15.5: A Succinct One-to-Many Data Model

15.8 Automatically creating primary keys

In the above example, we arbitrarily assigned Frank the primary key of 42. However when we are inserting millions or rows, it is nice to have the database automatically generate the values for the id column. We do this by declaring the `id` column as a `PRIMARY KEY` and leave out the `id` value when inserting the row:

```
DROP TABLE IF EXISTS Artist;
CREATE TABLE Artist (id INTEGER PRIMARY KEY,
    name TEXT, eyes TEXT);
INSERT INTO Artist (name, eyes)
   VALUES ('Frank Sinatra', 'blue');
```

Now we have instructed the database to auto-assign us a unique value to the Frank Sinatra row. But we then need a way to have the database tell us the `id` value for the recently inserted row. One way is to use a `SELECT` statement to retrieve data from an SQLite built-in-function called `last_insert_rowid()`.

```
sqlite> DROP TABLE IF EXISTS Artist;
sqlite> CREATE TABLE Artist (id INTEGER PRIMARY KEY,
   ...>     name TEXT, eyes TEXT);
sqlite> INSERT INTO Artist (name, eyes)
   ...>    VALUES ('Frank Sinatra', 'blue');
sqlite> select last_insert_rowid();
1
sqlite> SELECT * FROM Artist;
1|Frank Sinatra|blue
sqlite>
```

Once we know the `id` of our 'Frank Sinatra' row, we can use it when we `INSERT` the tracks into the `Track` table. As a general strategy, we add these `id` columns to any table we create:

```
sqlite> DROP TABLE IF EXISTS Track;
sqlite> CREATE TABLE Track (id INTEGER PRIMARY KEY,
   ...>     title TEXT, plays INTEGER, artist_id INTEGER);
```

Note that the `artist_id` value is the new auto-assigned row in the `Artist` table and that while we added an `INTEGER PRIMARY KEY` to the `Track` table, we did not include `id` in the list of fields on the `INSERT` statements into the `Track` table. Again this tells the database to choose a unique value for us for the `id` column.

```
sqlite> INSERT INTO Track (title, plays, artist_id)
   ...>     VALUES ('My Way', 15, 1);
sqlite> select last_insert_rowid();
1
sqlite> INSERT INTO Track (title, plays, artist_id)
   ...>     VALUES ('New York', 25, 1);
sqlite> select last_insert_rowid();
2
sqlite>
```

You can call `SELECT last_insert_rowid();` after each of the inserts to retrieve the value that the database assigned to the `id` of each newly created row. Later when we are coding in Python, we can ask for the `id` value in our code and store it in a variable for later use.

15.9 Logical keys for fast lookup

If we had a table full of artists and a table full of tracks, each with a foreign key link to a row in a table full of artists and we wanted to list all the tracks that were sung by 'Frank Sinatra' as follows:

```
SELECT title, plays, name, eyes
FROM Track JOIN Artist
ON Track.artist_id = Artist.id
WHERE Artist.name = 'Frank Sinatra';
```

Since we have two tables and a foreign key between the two tables, our data is well-modeled, but if we are going to have millions of records in the `Artist` table and going to do a lot of lookups by artist name, we would benefit if we gave the database a hint about our intended use of the `name` column.

We do this by adding an "index" to a text column that we intend to use in `WHERE` clauses:

```
CREATE INDEX artist_name ON Artist(name);
```

When the database has been told that an index is needed on a column in a table, it stores extra information to make it possible to look up a row more quickly using the indexed field (`name` in this example). Once you request that an index be created, there is nothing special that is needed in the SQL to access the table. The database keeps the index up to date as data is inserted, deleted, and updated, and uses it automatically if it will increase the performance of a database query.

These text columns that are used to find rows based on some information in the "real world" like the name of an artist are called *Logical keys*.

15.10 Adding constraints to the database

We can also use an index to enforce a constraint (i.e. rules) on our database operations. The most common constraint is a *uniqueness constraint* which insists

that all of the values in a column are unique. We can add the optional `UNIQUE` keyword, to the `CREATE INDEX` statement to tell the database that we would like it to enforce the constraint on our SQL. We can drop and re-create the `artist_name` index with a `UNIQUE` constraint as follows.

```
DROP INDEX artist_name;
CREATE UNIQUE INDEX artist_name ON Artist(name);
```

If we try to insert 'Frank Sinatra' a second time, it will fail with an error.

```
sqlite> SELECT * FROM Artist;
1|Frank Sinatra|blue
sqlite> INSERT INTO Artist (name, eyes)
   ...>    VALUES ('Frank Sinatra', 'blue');
Runtime error: UNIQUE constraint failed: Artist.name (19)
sqlite>
```

We can tell the database to ignore any duplicate key errors by adding the `IGNORE` keyword to the `INSERT` statement as follows:

```
sqlite> INSERT OR IGNORE INTO Artist (name, eyes)
   ...>     VALUES ('Frank Sinatra', 'blue');
sqlite> SELECT id FROM Artist WHERE name='Frank Sinatra';
1
sqlite>
```

By combining an `INSERT OR IGNORE` and a `SELECT` we can insert a new record if the name is not already there and whether or not the record is already there, retrieve the *primary* key of the record.

```
sqlite> INSERT OR IGNORE INTO Artist (name, eyes)
   ...>      VALUES ('Elvis', 'blue');
sqlite> SELECT id FROM Artist WHERE name='Elvis';
2
sqlite> SELECT * FROM Artist;
1|Frank Sinatra|blue
2|Elvis|blue
sqlite>
```

Since we have not added a uniqueness constraint to the eye color column, there is no problem having multiple 'Blue' values in the `eye` column.

15.11 Sample multi-table application

A sample application called `tracks_csv.py` shows how these ideas can be combined to parse textual data and load it into several tables using a proper data model with relational connections between the tables.

This application reads and parses a comma-separated file `tracks.csv` based on an export from Dr. Chuck's iTunes library.

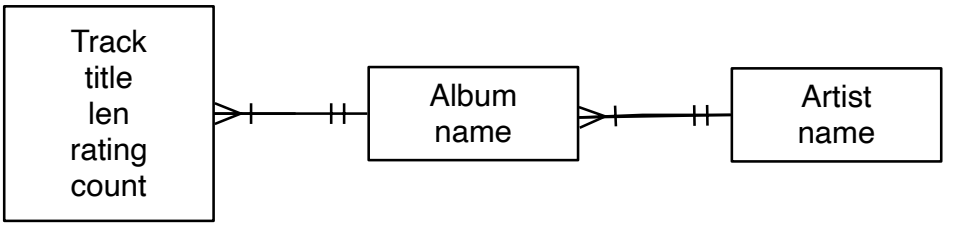

Figure 15.6: Tracks, Albums, and Artists

```
Another One Bites The Dust,Queen,Greatest Hits,55,100,217103
Asche Zu Asche,Rammstein,Herzeleid,79,100,231810
Beauty School Dropout,Various,Grease,48,100,239960
Black Dog,Led Zeppelin,IV,109,100,296620
...
```

The columns in this file are: title, artist, album, number of plays, rating (0-100) and length in milliseconds.

Our data model is shown in Figure 15.6 and described in SQL as follows:

```
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Album;
DROP TABLE IF EXISTS Track;

CREATE TABLE Artist (
    id INTEGER PRIMARY KEY,
    name TEXT UNIQUE
);

CREATE TABLE Album (
    id INTEGER PRIMARY KEY,
    artist_id  INTEGER,
    title TEXT UNIQUE
);

CREATE TABLE Track (
    id INTEGER PRIMARY KEY,
    title TEXT UNIQUE,
    album_id INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
);
```

We are adding the `UNIQUE` keyword to `TEXT` columns that we would like to have a uniqueness constraint that we will use in `INSERT IGNORE` statements. This is more succinct than separate `CREATE INDEX` statements but has the same effect.

With these tables in place, we write the following code `tracks_csv.py` to parse the data and insert it into the tables:

```
import sqlite3
```

```
conn = sqlite3.connect('trackdb.sqlite')
cur = conn.cursor()

handle = open('tracks.csv')

for line in handle:
    line = line.strip();
    pieces = line.split(',')
    if len(pieces) != 6 : continue

    name = pieces[0]
    artist = pieces[1]
    album = pieces[2]
    count = pieces[3]
    rating = pieces[4]
    length = pieces[5]

    print(name, artist, album, count, rating, length)

    cur.execute('''INSERT OR IGNORE INTO Artist (name)
        VALUES ( ? )''', ( artist, ) )
    cur.execute('SELECT id FROM Artist WHERE name = ? ', (artist, ))
    artist_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Album (title, artist_id)
        VALUES ( ?, ? )''', ( album, artist_id ) )
    cur.execute('SELECT id FROM Album WHERE title = ? ', (album, ))
    album_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Track
        (title, album_id, len, rating, count)
        VALUES ( ?, ?, ?, ?, ? )''',
        ( name, album_id, length, rating, count ) )

    conn.commit()
```

You can see that we are repeating the pattern of `INSERT OR IGNORE` followed by a `SELECT` to get the appropriate `artist_id` and `album_id` for use in later `INSERT` statements. We start from `Artist` because we need `artist_id` to insert the `Album` and need the `album_id` to insert the `Track`.

If we look at the `Album` table, we can see that the entries were added and assigned a *primary* key as necessary as the data was parsed. We can also see the *foreign key* pointing to a row in the `Artist` table for each `Album` row.

```
sqlite> .mode column
sqlite> SELECT * FROM Album LIMIT 5;
id  artist_id  title
--  ---------  ----------------
1   1          Greatest Hits
2   2          Herzeleid
```

```
3   3          Grease
4   4          IV
5   5          The Wall [Disc 2]
```

We can reconstruct all of the `Track` data, following all the relations using `JOIN / ON` clauses. You can see both ends of each of the (2) relational connections in each row in the output below:

```
sqlite> .mode line
sqlite> SELECT * FROM Track
   ...> JOIN Album ON Track.album_id = Album.id
   ...> JOIN Artist ON Album.artist_id = Artist.id
   ...> LIMIT 2;
       id = 1
    title = Another One Bites The Dust
 album_id = 1
      len = 217103
   rating = 100
    count = 55
       id = 1
artist_id = 1
    title = Greatest Hits
       id = 1
     name = Queen

       id = 2
    title = Asche Zu Asche
 album_id = 2
      len = 231810
   rating = 100
    count = 79
       id = 2
artist_id = 2
    title = Herzeleid
       id = 2
     name = Rammstein
```

This example shows three tables and two *one-to-many* relationships between the tables. It also shows how to use indexes and uniqueness constraints to programmatically construct the tables and their relationships.

https://en.wikipedia.org/wiki/One-to-many_(data_model)

Up next we will look at the many-to-many relationships in data models.

15.12 Many to many relationships in databases

Some data relationships cannot be modeled by a simple one-to-many relationship. For example, lets say we are going to build a data model for a course management system. There will be courses, users, and rosters. A user can be on the roster for many courses and a course will have many users on its roster.

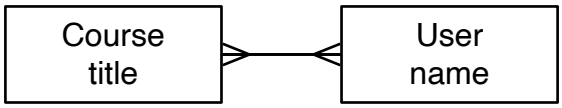

Figure 15.7: A Many to Many Relationship

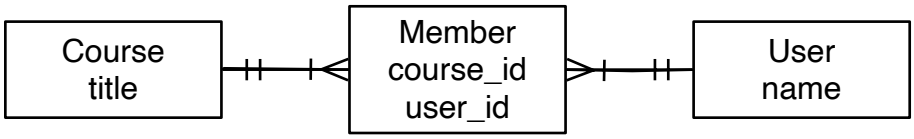

Figure 15.8: A Many to Many Connector Table

It is pretty simple to *draw* a many-to-many relationship as shown in Figure 15.7. We simply draw two tables and connect them with a line that has the "many" indicator on both ends of the lines. The problem is how to *implement* the relationship using primary keys and foreign keys.

Before we explore how we implement many-to-many relationships, let's see if we could hack something up by extending a one-to many relationship.

If SQL supported the notion of arrays, we might try to define this:

```
CREATE TABLE Course (
    id      INTEGER PRIMARY KEY,
    title   TEXT UNIQUE
    student_ids ARRAY OF INTEGER;
);
```

Sadly, while this is a tempting idea, SQL does not support arrays.[3]

Or we could just make long string and concatenate all the `User` primary keys into a long string separated by commas.

```
CREATE TABLE Course (
    id      INTEGER PRIMARY KEY,
    title   TEXT UNIQUE
    student_ids ARRAY OF INTEGER;
);

INSERT INTO Course (title, student_ids)
VALUES( 'si311', '1,3,4,5,6,9,14');
```

This would be very inefficient because as the course roster grows in size and the number of courses increases it becomes quite expensive to figure out which courses have student 14 on their roster.

[3]Some SQL dialects support arrays but arrays do not scale well. NoSQL databases use arrays and data replication but at a cost of database integrity. NoSQL is a story for another course https://www.pg4e.com/

Instead of either of these approaches, we model a many-to-many relationship using an additional table that we call a "junction table", "through table", "connector table", or "join table" as shown in Figure 15.8. The purpose of this table is to capture the *connection* between *a* course and *a* student.

In a sense the table sits between the `Course` and `User` table and has a one-to-many relationship to both tables. By using an intermediate table we break a many-to-many relationship into two one-to-many relationships. Databases are very good at modeling and processing one-to-many relationships.

An example `Member` table would be as follows:

```
CREATE TABLE User (
    id     INTEGER PRIMARY KEY,
    name   TEXT UNIQUE
);

CREATE TABLE Course (
    id     INTEGER PRIMARY KEY,
    title  TEXT UNIQUE
);

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    PRIMARY KEY (user_id, course_id)
);
```

Following our naming convention, `Member.user_id` and `Member.course_id` are foreign keys pointing at the corresponding rows in the `User` and `Course` tables. Each entry in the member table links a row in the `User` table to a row in the `Course` table by going *through* the `Member` table.

We indicate that the *combination* of `course_id` and `user_id` is the `PRIMARY KEY` for the `Member` table, also creating an uniqueness constraint for a `course_id` / `user_id` combination.

Now lets say we need to insert a number of students into the rosters of a number of courses. Lets assume the data comes to us in a JSON-formatted file with records like this:

```
[
  [ "Charley", "si110"],
  [ "Mea", "si110"],
  [ "Hattie", "si110"],
  [ "Keziah", "si110"],
  [ "Rosa", "si106"],
  [ "Mea", "si106"],
  [ "Mairin", "si106"],
  [ "Zendel", "si106"],
  [ "Honie", "si106"],
  [ "Rosa", "si106"],
...
]
```

We could write code as follows to read the JSON file and insert the members of each course roster into the database using the following code:

```
import json
import sqlite3

conn = sqlite3.connect('rosterdb.sqlite')
cur = conn.cursor()

str_data = open('roster_data_sample.json').read()
json_data = json.loads(str_data)

for entry in json_data:

    name = entry[0]
    title = entry[1]

    print((name, title))

    cur.execute('''INSERT OR IGNORE INTO User (name)
        VALUES ( ? )''', ( name, ) )
    cur.execute('SELECT id FROM User WHERE name = ? ', (name, ))
    user_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Course (title)
        VALUES ( ? )''', ( title, ) )
    cur.execute('SELECT id FROM Course WHERE title = ? ', (title, ))
    course_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Member
        (user_id, course_id) VALUES ( ?, ? )''',
        ( user_id, course_id ) )

    conn.commit()
```

Like in a previous example, we first make sure that we have an entry in the `User` table and know the primary key of the entry as well as an entry in the `Course` table and know its primary key. We use the 'INSERT OR IGNORE' and 'SELECT' pattern so our code works regardless of whether the record is in the table or not.

Our insert into the `Member` table is simply inserting the two integers as a new or existing row depending on the constraint to make sure we do not end up with duplicate entries in the `Member` table for a particular `user_id` / `course_id` combination.

To reconstruct our data across all three tables, we again use `JOIN` / `ON` to construct a `SELECT` query;

```
sqlite> SELECT * FROM Course
   ...> JOIN Member ON Course.id = Member.course_id
   ...> JOIN User ON Member.user_id = User.id;
```

```
+----+-------+---------+-----------+----+---------+
| id | title | user_id | course_id | id |  name   |
+----+-------+---------+-----------+----+---------+
| 1  | si110 | 1       | 1         | 1  | Charley |
| 1  | si110 | 2       | 1         | 2  | Mea     |
| 1  | si110 | 3       | 1         | 3  | Hattie  |
| 1  | si110 | 4       | 1         | 4  | Lyena   |
| 1  | si110 | 5       | 1         | 5  | Keziah  |
| 1  | si110 | 6       | 1         | 6  | Ellyce  |
| 1  | si110 | 7       | 1         | 7  | Thalia  |
| 1  | si110 | 8       | 1         | 8  | Meabh   |
| 2  | si106 | 2       | 2         | 2  | Mea     |
| 2  | si106 | 10      | 2         | 10 | Mairin  |
| 2  | si106 | 11      | 2         | 11 | Zendel  |
| 2  | si106 | 12      | 2         | 12 | Honie   |
| 2  | si106 | 9       | 2         | 9  | Rosa    |
+----+-------+---------+-----------+----+---------+
sqlite>
```

You can see the three tables from left to right - `Course`, `Member`, and `User` and you can see the connections between the primary keys and foreign keys in each row of output.

15.13 Modeling data at the many-to-many connection

While we have presented the "join table" as having two foreign keys making a connection between rows in two tables, this is the simplest form of a join table. It is quite common to want to add some data to the connection itself.

Continuing with our example of users, courses, and rosters to model a simple learning management system, we will also need to understand the *role* that each user is assigned in each course.

If we first try to solve this by adding an "instructor" flag to the `User` table, we will find that this does not work because a user can be a instructor in one course and a student in another course. If we add an `instructor_id` to the `Course` table it will not work because a course can have multiple instructors. And there is no one-to-many hack that can deal with the fact that the number of roles will expand into roles like Teaching Assistant or Parent.

But if we simply add a `role` column to the `Member` table - we can represent a wide range of roles, role combinations, etc.

Lets change our member table as follows:

```
DROP TABLE Member;

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role        INTEGER,
```

```
    PRIMARY KEY (user_id, course_id)
);
```

Para sa simplicity, magde-decide tayo na zero sa role ay nangangahulugang "student" at isa sa `role` ay nangangahulugang instructor. Ipagpalagay natin na ang JSON data natin ay na-augment ng role tulad ng sumusunod:

```
[
  [ "Charley", "si110", 1],
  [ "Mea", "si110", 0],
  [ "Hattie", "si110", 0],
  [ "Keziah", "si110", 0],
  [ "Rosa", "si106", 0],
  [ "Mea", "si106", 1],
  [ "Mairin", "si106", 0],
  [ "Zendel", "si106", 0],
  [ "Honie", "si106", 0],
  [ "Rosa", "si106", 0],
...
]
```

Maaari nating baguhin ang program na `roster.py` sa itaas para isama ang role tulad ng sumusunod:

```
for entry in json_data:

    name = entry[0]
    title = entry[1]
    role = entry[2]

    ...

    cur.execute('''INSERT OR REPLACE INTO Member
        (user_id, course_id, role) VALUES ( ?, ?, ? )''',
        ( user_id, course_id, role ) )
```

Sa tunay na system, malamang gagawa tayo ng `Role` table at gawin ang `role` column sa `Member` na foreign key sa Role table tulad ng sumusunod:

```
DROP TABLE Member;

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role_id     INTEGER,
    PRIMARY KEY (user_id, course_id, role_id)
);

CREATE TABLE Role (
    id          INTEGER PRIMARY KEY,
```

```
    name        TEXT UNIQUE
);

INSERT INTO Role (id, name) VALUES (0, 'Student');
INSERT INTO Role (id, name) VALUES (1, 'Instructor');
```

Notice that because we declared the `id` column in the `Role` table as a `PRIMARY KEY`, we *could* omit it in the `INSERT` statement. But we can also choose the `id` value as long as the value is not already in the `id` column and does not violate the implied `UNIQUE` constaint on primary keys.

15.14 Summary

This chapter has covered a lot of ground to give you an overview of the basics of using a database in Python. It is more complicated to write the code to use a database to store data than Python dictionaries or flat files so there is little reason to use a database unless your application truly needs the capabilities of a database. The situations where a database can be quite useful are: (1) when your application needs to make many small random updates within a large data set, (2) when your data is so large it cannot fit in a dictionary and you need to look up information repeatedly, or (3) when you have a long-running process that you want to be able to stop and restart and retain the data from one run to the next.

You can build a simple database with a single table to suit many application needs, but most problems will require several tables and links/relationships between rows in different tables. When you start making links between tables, it is important to do some thoughtful design and follow the rules of database normalization to make the best use of the database's capabilities. Since the primary motivation for using a database is that you have a large amount of data to deal with, it is important to model your data efficiently so your programs run as fast as possible.

15.15 Debugging

Ang isang karaniwang pattern kapag gumagawa ka ng Python program para kumonekta sa SQLite database ay patakbuhin ang Python program at suriin ang results gamit ang Database Browser for SQLite. Ang browser ay nagpapahintulot sa iyo na mabilis na suriin kung gumagana nang maayos ang program mo.

Dapat kang maging maingat dahil ang SQLite ay nag-aalaga para pigilan ang dalawang programs na baguhin ang parehong data nang sabay. Halimbawa, kung bubuksan mo ang database sa browser at gumawa ng pagbabago sa database at hindi pa na-press ang "save" button sa browser, ang browser ay "naglo-lock" ng database file at pumipigil sa anumang iba pang program na ma-access ang file. Sa partikular, ang Python program mo ay hindi makaka-access sa file kung naka-lock ito.

Kaya ang solusyon ay siguraduhing isara ang database browser o gamitin ang *File* menu para isara ang database sa browser bago subukang ma-access ang database mula sa Python para maiwasan ang problema ng Python code mo na mabigo dahil naka-lock ang database.

15.16 Glossary

attribute Isa sa mga values sa loob ng tuple. Mas karaniwang tinatawag na "column" o "field".

constraint Kapag sinasabi natin sa database na ipatupad ang rule sa field o row sa table. Ang karaniwang constraint ay pagpilit na hindi maaaring magkaroon ng duplicate values sa partikular na field (i.e., lahat ng values ay dapat unique).

cursor Ang cursor ay nagpapahintulot sa iyo na mag-execute ng SQL commands sa database at kumuha ng data mula sa database. Ang cursor ay katulad ng socket o file handle para sa network connections at files, ayon sa pagkakabanggit.

database browser Piraso ng software na nagpapahintulot sa iyo na direktang kumonekta sa database at manipulahin ang database nang direkta nang hindi sumusulat ng program.

foreign key Numeric key na tumuturo sa primary key ng row sa iba pang table. Ang foreign keys ay nagtatatag ng relationships sa pagitan ng rows na nakastore sa iba't ibang tables.

index Karagdagang data na nagma-maintain ang database software bilang rows at nag-i-insert sa table para gawing napakabilis ang lookups.

logical key Key na ginagamit ng "outside world" para maghanap ng partikular na row. Para sa halimbawa sa table ng user accounts, ang email address ng tao ay maaaring maging magandang kandidato bilang logical key para sa data ng user.

normalization Pagde-design ng data model para walang data na na-replicate. Nag-i-store tayo ng bawat item ng data sa isang lugar sa database at nagre-reference dito sa ibang lugar gamit ang foreign key.

primary key Numeric key na na-a-assign sa bawat row na ginagamit para tumukoy sa isang row sa table mula sa iba pang table. Kadalasan ang database ay naka-configure para awtomatikong mag-assign ng primary keys habang na-i-insert ang rows.

relation Area sa loob ng database na naglalaman ng tuples at attributes. Mas karaniwang tinatawag na "table".

tuple Isang entry sa database table na set ng attributes. Mas karaniwang tinatawag na "row".

Chapter 16

Visualizing data

Hanggang ngayon natututo tayo ng Python language at pagkatapos natututo kung paano gamitin ang Python, network, at databases para manipulahin ang data.

Sa chapter na ito, titingnan natin ang tatlong kumpletong applications na pinagsasama ang lahat ng mga bagay na ito para pamahalaan at i-visualize ang data. Maaari mong gamitin ang mga applications na ito bilang sample code para matulungan kang magsimula sa pagsolusyon ng real-world problem.

Ang bawat isa sa applications ay ZIP file na maaari mong i-download at i-extract sa computer mo at i-execute.

16.1 Building a OpenStreetMap from geocoded data

Sa project na ito, gumagamit tayo ng OpenStreetMap geocoding API para linisin ang ilang user-entered geographic locations ng university names at pagkatapos ilagay ang data sa aktwal na OpenStreetMap.

Para magsimula, i-download ang application mula sa:

www.py4e.com/code3/opengeo.zip

Ang unang problema na solusyonan ay ang mga geocoding APIs na ito ay rate-limited sa tiyak na bilang ng requests bawat araw. Kung mayroon kang maraming data, maaaring kailangan mong huminto at muling simulan ang lookup process nang ilang beses. Kaya hinahati natin ang problema sa dalawang phases.

Sa unang phase kinukuha natin ang input "survey" data natin sa file *where.data* at binabasa ito isang linya sa isang pagkakataon, at kumukuha ng geocoded information mula sa Google at nag-i-store nito sa database *geodata.sqlite*. Bago gamitin ang geocoding API para sa bawat user-entered location, simpleng sinusuri natin kung mayroon na tayong data para sa partikular na linya ng input. Ang database ay gumagana bilang local "cache" ng geocoding data natin para siguraduhin na hindi tayo kailanman hihingi sa Google ng parehong data nang dalawang beses.

Figure 16.1: An OpenStreetMap

Maaari mong muling simulan ang proseso anumang oras sa pamamagitan ng pagtanggal ng file *geodata.sqlite*.

Patakbuhin ang *geoload.py* program. Ang program na ito ay magbabasa ng input lines sa *where.data* at para sa bawat linya susuriin kung nasa database na ito. Kung wala tayong data para sa location, tatawagin nito ang geocoding API para kunin ang data at i-store ito sa database.

Narito ang sample run pagkatapos mayroon nang ilang data sa database:

```
Found in database AGH University of Science and Technology

Found in database Academy of Fine Arts Warsaw Poland

Found in database American University in Cairo

Found in database Arizona State University

Found in database Athens Information Technology

Retrieving https://py4e-data.dr-chuck.net/
   opengeo?q=BITS+Pilani
Retrieved 794 characters {"type":"FeatureColl

Retrieving https://py4e-data.dr-chuck.net/
   opengeo?q=Babcock+University
Retrieved 760 characters {"type":"FeatureColl

Retrieving https://py4e-data.dr-chuck.net/
   opengeo?q=Banaras+Hindu+University
Retrieved 866 characters {"type":"FeatureColl
...
```

Ang unang limang locations ay nasa database na at kaya sila ay na-skip. Ang program ay nag-scan hanggang sa punto kung saan nakakahanap ito ng bagong locations at nagsisimulang kunin ang mga ito.

Ang *geoload.py* program ay maaaring itigil anumang oras, at mayroong counter na maaari mong gamitin para limitahan ang bilang ng tawag sa geocoding API para sa bawat run. Dahil ang *where.data* ay may ilang daang data items lang, hindi ka dapat makakaranas ng daily rate limit, pero kung mayroon kang mas maraming data maaaring kailangan ng ilang runs sa loob ng ilang araw para magkaroon ang database mo ng lahat ng geocoded data para sa input mo.

Kapag mayroon ka nang ilang data na na-load sa *geodata.sqlite*, maaari mong i-visualize ang data gamit ang *geodump.py* program. Ang program na ito ay nagbabasa ng database at sumusulat ng file *where.js* na may location, latitude, at longitude sa form ng executable JavaScript code.

Ang run ng *geodump.py* program ay ganito:

```
AGH University of Science and Technology, Czarnowiejska,
Czarna Wie's, Krowodrza, Krak'ow, Lesser Poland
Voivodeship, 31-126, Poland 50.0657 19.91895

Academy of Fine Arts, Krakowskie Przedmie'scie,
Northern 'Sr'odmie'scie, 'Sr'odmie'scie, Warsaw, Masovian
Voivodeship, 00-046, Poland 52.239 21.0155
...
260 lines were written to where.js
Open the where.html file in a web browser to view the data.
```

Ang file na *where.html* ay binubuo ng HTML at JavaScript para i-visualize ang Google map. Binabasa nito ang pinakabagong data sa *where.js* para makuha ang data na i-visualize. Narito ang format ng file na *where.js*:

```
myData = [
[50.0657,19.91895,
'AGH University of Science and Technology, Czarnowiejska,
Czarna Wie's, Krowodrza, Krak'ow, Lesser Poland
Voivodeship, 31-126, Poland '],
[52.239,21.0155,
'Academy of Fine Arts, Krakowskie Przedmie'sciee,
'Sr'odmie'scie P'olnocne, 'Sr'odmie'scie, Warsaw,
Masovian Voivodeship, 00-046, Poland'],
   ...
];
```

Ito ay JavaScript variable na naglalaman ng list ng lists. Ang syntax para sa JavaScript list constants ay napakatulad sa Python, kaya ang syntax ay dapat pamilyar sa iyo.

Simpleng buksan ang *where.html* sa browser para makita ang locations. Maaari mong i-hover sa bawat map pin para hanapin ang location na ibinalik ng geocoding API para sa user-entered input. Kung hindi mo makita ang anumang data kapag binubuksan mo ang file na *where.html*, maaaring gusto mong suriin ang JavaScript o developer console para sa browser mo.

Figure 16.2: A Page Ranking

16.2 Visualizing networks and interconnections

Sa application na ito, gagawa tayo ng ilang functions ng search engine. Una tayong mag-spider ng maliit na subset ng web at magpatakbo ng simplified version ng Google page rank algorithm para matukoy kung alin ang pages na pinakamataas ang connectivity, at pagkatapos i-visualize ang page rank at connectivity ng maliit na sulok natin ng web. Gagamitin natin ang D3 JavaScript visualization library http://d3js.org/ para gumawa ng visualization output.

Maaari mong i-download at i-extract ang application na ito mula sa:

www.py4e.com/code3/pagerank.zip

Ang unang program (*spider.py*) ay nag-crawl ng web site at kumukuha ng serye ng pages sa database (*spider.sqlite*), na nagre-record ng links sa pagitan ng pages. Maaari mong muling simulan ang proseso anumang oras sa pamamagitan ng pagtanggal ng file na *spider.sqlite* at muling pagpatakbo ng *spider.py*.

```
Enter web url or enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
How many pages:2
1 http://www.dr-chuck.com/ 12
2 http://www.dr-chuck.com/csev-blog/ 57
How many pages:
```

Sa sample run na ito, sinabi natin sa kanya na i-crawl ang website at kunin ang dalawang pages. Kung muling patakbuhin mo ang program at sabihin sa kanya

na i-crawl ang mas maraming pages, hindi ito magre-crawl ng anumang pages na nasa database na. Sa pag-restart pumupunta ito sa random na non-crawled page at nagsisimula doon. Kaya ang bawat sunud-sunod na run ng *spider.py* ay additive.

```
Enter web url or enter: http://www.dr-chuck.com/
['http://www.dr-chuck.com']
How many pages:3
3 http://www.dr-chuck.com/csev-blog 57
4 http://www.dr-chuck.com/dr-chuck/resume/speaking.htm 1
5 http://www.dr-chuck.com/dr-chuck/resume/index.htm 13
How many pages:
```

Maaari kang magkaroon ng maraming starting points sa parehong database-sa loob ng program, ang mga ito ay tinatawag na "webs". Ang spider ay pumipili nang random sa lahat ng non-visited links sa lahat ng webs bilang susunod na page na i-spider.

Kung gusto mong i-dump ang contents ng file na *spider.sqlite*, maaari mong patakbuhin ang *spdump.py* tulad ng sumusunod:

```
(5, None, 1.0, 3, 'http://www.dr-chuck.com/csev-blog')
(3, None, 1.0, 4, 'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, None, 1.0, 2, 'http://www.dr-chuck.com/csev-blog/')
(1, None, 1.0, 5, 'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
4 rows.
```

Ipinapakita nito ang bilang ng incoming links, ang lumang page rank, ang bagong page rank, ang id ng page, at ang url ng page. Ang program na *spdump.py* ay nagpapakita lang ng pages na may hindi bababa sa isang incoming link sa kanila.

Kapag mayroon ka nang ilang pages sa database, maaari mong patakbuhin ang page rank sa pages gamit ang program na *sprank.py*. Simpleng sinasabi mo lang sa kanya kung ilang page rank iterations ang patakbuhin.

```
How many iterations:2
1 0.546848992536
2 0.226714939664
[(1, 0.559), (2, 0.659), (3, 0.985), (4, 2.135), (5, 0.659)]
```

Maaari mong i-dump ang database ulit para makita na na-update na ang page rank:

```
(5, 1.0, 0.985, 3, 'http://www.dr-chuck.com/csev-blog')
(3, 1.0, 2.135, 4, 'http://www.dr-chuck.com/dr-chuck/resume/speaking.htm')
(1, 1.0, 0.659, 2, 'http://www.dr-chuck.com/csev-blog/')
(1, 1.0, 0.659, 5, 'http://www.dr-chuck.com/dr-chuck/resume/index.htm')
4 rows.
```

Maaari mong patakbuhin ang *sprank.py* nang maraming beses hangga't gusto mo at ito ay simpleng magre-refine ng page rank sa bawat pagkakataon na patakbuhin mo ito. Maaari mo ring patakbuhin ang *sprank.py* nang ilang beses at pagkatapos

mag-spider ng ilang higit pang pages gamit ang *spider.py* at pagkatapos patakbuhin ang *sprank.py* para muling mag-converge ang page rank values. Ang search engine ay karaniwang nagpapatakbo ng parehong crawling at ranking programs sa lahat ng oras.

Kung gusto mong muling simulan ang page rank calculations nang hindi muling nag-spider ng web pages, maaari mong gamitin ang *spreset.py* at pagkatapos muling simulan ang *sprank.py*.

```
How many iterations:50
1 0.546848992536
2 0.226714939664
3 0.0659516187242
4 0.0244199333
5 0.0102096489546
6 0.00610244329379
...
42 0.000109076928206
43 9.91987599002e-05
44 9.02151706798e-05
45 8.20451504471e-05
46 7.46150183837e-05
47 6.7857770908e-05
48 6.17124694224e-05
49 5.61236959327e-05
50 5.10410499467e-05
[(512, 0.0296), (1, 12.79), (2, 28.93), (3, 6.808), (4, 13.46)]
```

Para sa bawat iteration ng page rank algorithm nagpi-print ito ng average change sa page rank per page. Ang network sa simula ay medyo hindi balanse at kaya ang individual page rank values ay nagbabago nang malaki sa pagitan ng iterations. Pero sa ilang maikling iterations, ang page rank ay nagco-converge. Dapat mong patakbuhin ang *sprank.py* nang sapat na tagal para ang page rank values ay mag-converge.

Kung gusto mong i-visualize ang kasalukuyang top pages sa terms ng page rank, patakbuhin ang *spjson.py* para basahin ang database at isulat ang data para sa pinakamataas na linked pages sa JSON format para makita sa web browser.

```
Creating JSON output on spider.json...
How many nodes? 30
Open force.html in a browser to view the visualization
```

Maaari mong tingnan ang data na ito sa pamamagitan ng pagbubukas ng file na *force.html* sa web browser mo. Ipinapakita nito ang automatic layout ng nodes at links. Maaari mong i-click at i-drag ang anumang node at maaari mo ring i-double-click ang node para hanapin ang URL na kinakatawan ng node.

Kung muling patakbuhin mo ang iba pang utilities, muling patakbuhin ang *spjson.py* at pindutin ang refresh sa browser para makuha ang bagong data mula sa *spider.json*.

Figure 16.3: A Word Cloud from the Sakai Developer List

16.3 Visualizing mail data

Hanggang sa puntong ito sa libro, naging pamilyar ka na sa aming *mbox-short.txt* at *mbox.txt* data files. Ngayon panahon na para dalhin ang analysis natin ng email data sa susunod na level.

Sa totoong mundo, minsan kailangan mong kunin ang mail data mula sa servers. Maaaring tumagal ito ng ilang panahon at ang data ay maaaring hindi consistent, puno ng error, at nangangailangan ng maraming cleanup o adjustment. Sa section na ito, nagtatrabaho tayo sa application na pinakakumplikado hanggang ngayon at kumukuha ng halos gigabyte ng data at i-visualize ito.

Maaari mong i-download ang application na ito mula sa:

https://www.py4e.com/code3/gmane.zip

Gagamitin natin ang data mula sa libreng email list archiving service na tinatawag na *gmane* - ang service ay na-shut down na at para sa layunin ng course na ito, ang partial archive ay na-maintain sa http://mbox.dr-chuck.net. Ang gmane service ay napakapopular sa open source projects dahil nagbibigay ito ng magandang searchable archive ng kanilang email activity.

http://mbox.dr-chuck.net/export.php

Kapag ang Sakai email data ay na-spider gamit ang software na ito, gumawa ito ng halos Gigabyte ng data at tumagal ng ilang runs sa loob ng ilang araw. Ang file na *README.txt* sa ZIP sa itaas ay maaaring may instructions tungkol sa kung paano mo maaaring i-download ang pre-spidered copy ng file na *content.sqlite* para sa karamihan ng Sakai email corpus para hindi mo kailangang mag-spider ng limang

araw lang para patakbuhin ang programs. Kung i-download mo ang pre-spidered content, dapat mo pa ring patakbuhin ang spidering process para makahabol sa mas bagong messages.

Ang unang hakbang ay i-spider ang repository. Ang base URL ay hard-coded sa *gmane.py* at hard-coded sa Sakai developer list. Maaari mong i-spider ang iba pang repository sa pamamagitan ng pagbabago ng base url na iyon. Siguraduhing tanggalin ang file na *content.sqlite* kung magpapalit ka ng base url.

Ang file na *gmane.py* ay gumagana bilang responsible caching spider sa diwa na mabagal itong tumatakbo at kumukuha ng isang mail message bawat segundo para maiwasan ang ma-throttle. Nag-i-store ito ng lahat ng data nito sa database at maaaring ma-interrupt at muling simulan nang kasing dami ng kailangan. Maaari itong tumagal ng maraming oras para kunin ang lahat ng data. Kaya maaaring kailangan mong muling simulan nang ilang beses.

Narito ang run ng *gmane.py* na kumukuha ng huling limang messages ng Sakai developer list:

```
How many messages:10
http://mbox.dr-chuck.net/sakai.devel/51410/51411 9460
    nealcaidin@sakaifoundation.org 2013-04-05 re: [building ...
http://mbox.dr-chuck.net/sakai.devel/51411/51412 3379
    samuelgutierrezjimenez@gmail.com 2013-04-06 re: [building ...
http://mbox.dr-chuck.net/sakai.devel/51412/51413 9903
    da1@vt.edu 2013-04-05 [building sakai] melete 2.9 oracle ...
http://mbox.dr-chuck.net/sakai.devel/51413/51414 349265
    m.shedid@elraed-it.com 2013-04-07 [building sakai] ...
http://mbox.dr-chuck.net/sakai.devel/51414/51415 3481
    samuelgutierrezjimenez@gmail.com 2013-04-07 re: ...
http://mbox.dr-chuck.net/sakai.devel/51415/51416 0

Does not start with From
```

Ang program ay nag-scan ng *content.sqlite* mula sa isa hanggang sa unang message number na hindi pa na-spider at nagsisimulang mag-spider sa message na iyon. Nagpapatuloy ito sa pag-spider hanggang na-spider na nito ang gustong bilang ng messages o umabot ito sa page na hindi mukhang properly formatted message.

Minsan ang repository ay kulang ng message. Marahil ang administrators ay maaaring magtanggal ng messages o marahil nawawala sila. Kung ang spider mo ay huminto, at mukhang nakahit ito ng missing message, pumunta sa SQLite Manager at magdagdag ng row na may missing id na iiwan ang lahat ng iba pang fields na blank at muling simulan ang *gmane.py*. Aalisin nito ang stuck na spidering process at payagan itong magpatuloy. Ang mga empty messages na ito ay hindi papansinin sa susunod na phase ng proseso.

Ang isang magandang bagay ay kapag na-spider mo na ang lahat ng messages at mayroon ka na sa *content.sqlite*, maaari mong patakbuhin ang *gmane.py* ulit para makakuha ng bagong messages habang ipinapadala sila sa list.

Ang data ng *content.sqlite* ay medyo raw, na may inefficient data model, at hindi compressed. Ito ay sinasadya dahil nagpapahintulot ito sa iyo na tingnan ang *content.sqlite* sa SQLite Manager para i-debug ang mga problema sa spidering

process. Masamang ideya na patakbuhin ang anumang queries laban sa database na ito, dahil magiging napakabagal sila.

Ang pangalawang proseso ay patakbuhin ang program na *gmodel.py*. Ang program na ito ay nagbabasa ng raw data mula sa *content.sqlite* at gumagawa ng cleaned-up at well-modeled version ng data sa file *index.sqlite*. Ang file na ito ay mas maliit (kadalasan 10X mas maliit) kaysa sa *content.sqlite* dahil nagko-compress din ito ng header at body text.

Sa bawat pagkakataon na tumatakbo ang *gmodel.py* tinatanggal nito at muling ginagawa ang *index.sqlite*, na nagpapahintulot sa iyo na i-adjust ang parameters nito at i-edit ang mapping tables sa *content.sqlite* para i-tweak ang data cleaning process. Ito ay sample run ng *gmodel.py*. Nagpi-print ito ng linya sa bawat pagkakataon na 250 mail messages ay na-proseso para makita mo ang ilang progress na nangyayari, dahil ang program na ito ay maaaring tumakbo nang ilang panahon na nagpo-proseso ng halos Gigabyte ng mail data.

```
Loaded allsenders 1588 and mapping 28 dns mapping 1
1 2005-12-08T23:34:30-06:00 ggolden22@mac.com
251 2005-12-22T10:03:20-08:00 tpamsler@ucdavis.edu
501 2006-01-12T11:17:34-05:00 lance@indiana.edu
751 2006-01-24T11:13:28-08:00 vrajgopalan@ucmerced.edu
...
```

Ang program na *gmodel.py* ay nagha-handle ng ilang data cleaning tasks.

Ang domain names ay na-truncate sa dalawang levels para sa .com, .org, .edu, at .net. Ang iba pang domain names ay na-truncate sa tatlong levels. Kaya ang si.umich.edu ay nagiging umich.edu at ang caret.cam.ac.uk ay nagiging cam.ac.uk. Ang email addresses ay pinipilit din na maging lower case, at ang ilan sa @gmane.org address tulad ng sumusunod

```
arwhyte-63aXycvo3TyHXe+LvDLADg@public.gmane.org
```

ay na-convert sa tunay na address tuwing mayroong tumutugmang tunay na email address sa ibang lugar sa message corpus.

Sa database na *mapping.sqlite* mayroong dalawang tables na nagpapahintulot sa iyo na mag-map ng parehong domain names at individual email addresses na nagbabago sa buong buhay ng email list. Halimbawa, si Steve Githens ay gumamit ng sumusunod na email addresses habang nagpapalit ng trabaho sa buong buhay ng Sakai developer list:

```
s-githens@northwestern.edu
sgithens@cam.ac.uk
swgithen@mtu.edu
```

Maaari tayong magdagdag ng dalawang entries sa Mapping table sa *mapping.sqlite* para ang *gmodel.py* ay magma-map ng lahat ng tatlo sa isang address:

```
s-githens@northwestern.edu ->  swgithen@mtu.edu
sgithens@cam.ac.uk -> swgithen@mtu.edu
```

Maaari mo ring gumawa ng katulad na entries sa DNSMapping table kung mayroong maraming DNS names na gusto mong i-map sa isang DNS. Ang sumusunod mapping ay idinagdag sa Sakai data:

```
iupui.edu -> indiana.edu
```

para lahat ng accounts mula sa iba't ibang Indiana University campuses ay na-track nang magkasama.

Maaari mong muling patakbuhin ang *gmodel.py* nang paulit-ulit habang tinitingnan mo ang data, at magdagdag ng mappings para gawing mas malinis at mas malinis ang data. Kapag tapos ka na, magkakaroon ka ng magandang indexed version ng email sa *index.sqlite*. Ito ang file na gagamitin para gumawa ng data analysis. Gamit ang file na ito, ang data analysis ay talagang mabilis.

Ang una, pinakasimpleng data analysis ay matukoy "sino ang nagpadala ng pinakamaraming mail?" at "aling organization ang nagpadala ng pinakamaraming mail"? Ginagawa ito gamit ang *gbasic.py*:

```
How many to dump? 5
Loaded messages= 51330 subjects= 25033 senders= 1584

Top 5 Email list participants
steve.swinsburg@gmail.com 2657
azeckoski@unicon.net 1742
ieb@tfd.co.uk 1591
csev@umich.edu 1304
david.horwitz@uct.ac.za 1184

Top 5 Email list organizations
gmail.com 7339
umich.edu 6243
uct.ac.za 2451
indiana.edu 2258
unicon.net 2055
```

Tandaan kung gaano mas mabilis tumatakbo ang *gbasic.py* kumpara sa *gmane.py* o kahit na *gmodel.py*. Lahat sila ay nagtatrabaho sa parehong data, pero ang *gbasic.py* ay gumagamit ng compressed at normalized data sa *index.sqlite*. Kung mayroon kang maraming data na pamahalaan, ang multistep process tulad ng nasa application na ito ay maaaring tumagal ng kaunti para ma-develop, pero makakatipid ka ng maraming oras kapag talagang nagsimula ka nang mag-explore at i-visualize ang data mo.

Maaari kang gumawa ng simpleng visualization ng word frequency sa subject lines sa file na *gword.py*:

```
Range of counts: 33229 129
Output written to gword.js
```

Gumagawa ito ng file na *gword.js* na maaari mong i-visualize gamit ang *gword.htm* para gumawa ng word cloud na katulad ng isa sa simula ng section na ito.

Ang pangalawang visualization ay ginagawa ng *gline.py*. Ito ay nagko-compute ng email participation ng organizations sa paglipas ng panahon.

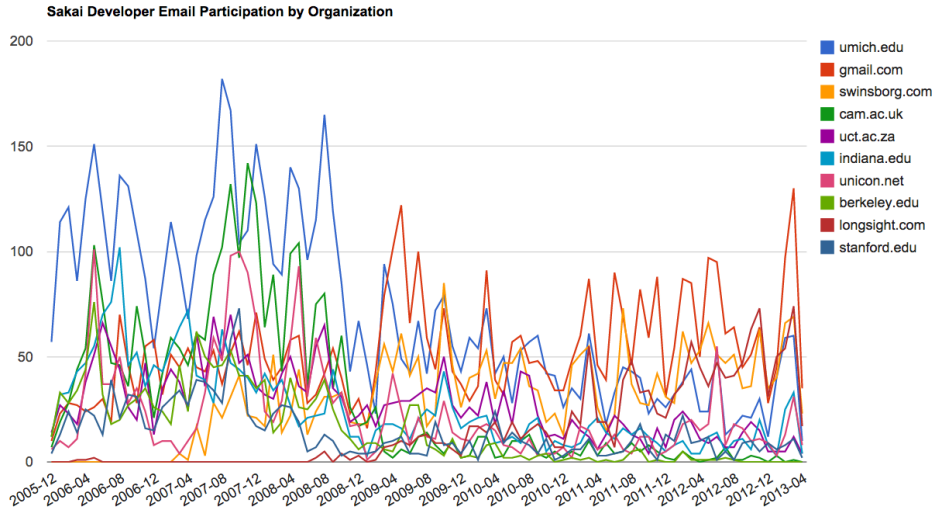

Figure 16.4: Sakai Mail Activity by Organization

```
Loaded messages= 51330 senders= 1584
Top 10 Oranizations
['gmail.com', 'umich.edu', 'uct.ac.za', 'indiana.edu',
'unicon.net', 'tfd.co.uk', 'berkeley.edu', 'longsight.com',
'stanford.edu', 'ox.ac.uk']
Output written to gline.js
```

Ang output nito ay isinusulat sa *gline.js* na na-visualize gamit ang *gline.htm*.

Ito ay relatively complex at sophisticated application at mayroong features para gumawa ng ilang tunay na data retrieval, cleaning, at visualization.

Appendix A

Contributions

A.1 Translations

Ang libro na ito ay na-translate sa ilang languages na makikita sa https://www.py4e.com/book

- Spanish - Python para todos: Explorando la información con Python 3 - Translated book, autograders, resources, at web site sa https://es.py4e.com Contributors: Juan Carlos Perez Castellanos, Juan Dougnac, Daniel Merino Echeverría, Jaime Bermeo Ramírez at Fernando Tardío.
- Italian Python per tutti: Esplorare dati con Python3 Contributors: Alessandro Rossetti at Vittore Zen
- Portuguese - Python Para Todos: Explorando Dados com Python 3 Translation: Yuri Loia de Medeiros
- Polish - Python dla wszystkich: Odkrywanie danych z Python 3 Translated book, autograders, resources, at web site sa https://py4e.pl Translation: Andrzej Wójtowicz (Adam Mickiewicz University sa Poznań, Poland)
- Greek - Translated book, autograders, resources, at web site sa https://gr.py4e.com Translation: Konstantia Kiourtidou
- Arabic - Translation: Electronics Go

Ang sinumang binigyan ng pahintulot na i-translate ang libro ay sumasang-ayon na magbigay ng libreng kopya ng libro online. Ang ilan sa mga translations ay available sa print at available para sa pagbili.

A.2 Contributor List for Python for Everybody

Ang source para sa libro ay na-maintain sa `github` at maaari mong makita ang maraming contributors para sa libro na naghahanap at nagmumungkahi ng fixes gamit ang pull requests.

https://github.com/csev/py4e/

Tingnan sa `Insights -> Contributors` para makita ang lahat ng mga tao na nag-ambag sa pamamagitan ng `github`.

Juan Carlos Perez Castellanos, Juan Dougnac, Daniel Merino Echeverría, Jaime Bermeo Ramírez, Fernando Tardío, Alessandro Rossetti, Vittore Zen, Yuri Loia de Medeiros, Konstantia Kiourtidou, Andrzej Wójtowicz, Elliott Hauser, Stephen Catto, Sue Blumenberg, Tamara Brunnock, Mihaela Mack, Chris Kolosiwsky, Dustin Farley, Jens Leerssen, Naveen KT, Mirza Ibrahimovic, Naveen (@togarnk), Zhou Fangyi, Alistair Walsh, Erica Brody, Jih-Sheng Huang, Louis Luangkesorn, and Michael Fudge

You can see contribution details at:

https://github.com/csev/py4e/graphs/contributors

A.3 Contributor List for Python for Informatics

Bruce Shields for copy editing early drafts, Sarah Hegge, Steven Cherry, Sarah Kathleen Barbarow, Andrea Parker, Radaphat Chongthammakun, Megan Hixon, Kirby Urner, Sarah Kathleen Barbrow, Katie Kujala, Noah Botimer, Emily Alinder, Mark Thompson-Kular, James Perry, Eric Hofer, Eytan Adar, Peter Robinson, Deborah J. Nelson, Jonathan C. Anthony, Eden Rassette, Jeannette Schroeder, Justin Feezell, Chuanqi Li, Gerald Gordinier, Gavin Thomas Strassel, Ryan Clement, Alissa Talley, Caitlin Holman, Yong-Mi Kim, Karen Stover, Cherie Edmonds, Maria Seiferle, Romer Kristi D. Aranas (RK), Grant Boyer, Hedemarrie Dussan,

A.4 Preface for "Think Python"

A.4.1 The strange history of "Think Python"

(Allen B. Downey)

In January 1999 I was preparing to teach an introductory programming class in Java. I had taught it three times and I was getting frustrated. The failure rate in the class was too high and, even for students who succeeded, the overall level of achievement was too low.

One of the problems I saw was the books. They were too big, with too much unnecessary detail about Java, and not enough high-level guidance about how to program. And they all suffered from the trap door effect: they would start out easy, proceed gradually, and then somewhere around Chapter 5 the bottom would fall out. The students would get too much new material, too fast, and I would spend the rest of the semester picking up the pieces.

Two weeks before the first day of classes, I decided to write my own book. My goals were:

- Keep it short. It is better for students to read 10 pages than not read 50 pages.

- Be careful with vocabulary. I tried to minimize the jargon and define each term at first use.
- Build gradually. To avoid trap doors, I took the most difficult topics and split them into a series of small steps.
- Focus on programming, not the programming language. I included the minimum useful subset of Java and left out the rest.

I needed a title, so on a whim I chose *How to Think Like a Computer Scientist.*

My first version was rough, but it worked. Students did the reading, and they understood enough that I could spend class time on the hard topics, the interesting topics and (most important) letting the students practice.

I released the book under the GNU Free Documentation License, which allows users to copy, modify, and distribute the book.

What happened next is the cool part. Jeff Elkner, a high school teacher in Virginia, adopted my book and translated it into Python. He sent me a copy of his translation, and I had the unusual experience of learning Python by reading my own book.

Jeff and I revised the book, incorporated a case study by Chris Meyers, and in 2001 we released *How to Think Like a Computer Scientist: Learning with Python*, also under the GNU Free Documentation License. As Green Tea Press, I published the book and started selling hard copies through Amazon.com and college book stores. Other books from Green Tea Press are available at greenteapress.com.

In 2003 I started teaching at Olin College and I got to teach Python for the first time. The contrast with Java was striking. Students struggled less, learned more, worked on more interesting projects, and generally had a lot more fun.

Over the last five years I have continued to develop the book, correcting errors, improving some of the examples and adding material, especially exercises. In 2008 I started work on a major revision—at the same time, I was contacted by an editor at Cambridge University Press who was interested in publishing the next edition. Good timing!

I hope you enjoy working with this book, and that it helps you learn to program and think, at least a little bit, like a computer scientist.

A.4.2 Acknowledgements for "Think Python"

(Allen B. Downey)

First and most importantly, I thank Jeff Elkner, who translated my Java book into Python, which got this project started and introduced me to what has turned out to be my favorite language.

I also thank Chris Meyers, who contributed several sections to *How to Think Like a Computer Scientist.*

And I thank the Free Software Foundation for developing the GNU Free Documentation License, which helped make my collaboration with Jeff and Chris possible.

I also thank the editors at Lulu who worked on *How to Think Like a Computer Scientist*.

I thank all the students who worked with earlier versions of this book and all the contributors (listed in an Appendix) who sent in corrections and suggestions.

And I thank my wife, Lisa, for her work on this book, and Green Tea Press, and everything else, too.

Allen B. Downey
Needham MA

Allen Downey is an Associate Professor of Computer Science at the Franklin W. Olin College of Engineering.

A.5 Contributor List for "Think Python"

(Allen B. Downey)

More than 100 sharp-eyed and thoughtful readers have sent in suggestions and corrections over the past few years. Their contributions, and enthusiasm for this project, have been a huge help.

For the detail on the nature of each of the contributions from these individuals, see the "Think Python" text.

Lloyd Hugh Allen, Yvon Boulianne, Fred Bremmer, Jonah Cohen, Michael Conlon, Benoit Girard, Courtney Gleason and Katherine Smith, Lee Harr, James Kaylin, David Kershaw, Eddie Lam, Man-Yong Lee, David Mayo, Chris McAloon, Matthew J. Moelter, Simon Dicon Montford, John Ouzts, Kevin Parks, David Pool, Michael Schmitt, Robin Shaw, Paul Sleigh, Craig T. Snydal, Ian Thomas, Keith Verheyden, Peter Winstanley, Chris Wrobel, Moshe Zadka, Christoph Zwerschke, James Mayer, Hayden McAfee, Angel Arnal, Tauhidul Hoque and Lex Berezhny, Dr. Michele Alzetta, Andy Mitchell, Kalin Harvey, Christopher P. Smith, David Hutchins, Gregor Lingl, Julie Peters, Florin Oprina, D. J. Webre, Ken, Ivo Wever, Curtis Yanko, Ben Logan, Jason Armstrong, Louis Cordier, Brian Cain, Rob Black, Jean-Philippe Rey at Ecole Centrale Paris, Jason Mader at George Washington University made a number Jan Gundtofte-Bruun, Abel David and Alexis Dinno, Charles Thayer, Roger Sperberg, Sam Bull, Andrew Cheung, C. Corey Capel, Alessandra, Wim Champagne, Douglas Wright, Jared Spindor, Lin Peiheng, Ray Hagtvedt, Torsten Hübsch, Inga Petuhhov, Arne Babenhauserheide, Mark E. Casida, Scott Tyler, Gordon Shephard, Andrew Turner, Adam Hobart, Daryl Hammond and Sarah Zimmerman, George Sass, Brian Bingham, Leah Engelbert-Fenton, Joe Funke, Chao-chao Chen, Jeff Paine, Lubos Pintes, Gregg Lind and Abigail Heithoff, Max Hailperin, Chotipat Pornavalai, Stanislaw Antol, Eric Pashman, Miguel Azevedo, Jianhua Liu, Nick King, Martin Zuther, Adam Zimmerman, Ratnakar Tiwari, Anurag Goel, Kelli Kratzer, Mark Griffiths, Roydan Ongie, Patryk Wolowiec, Mark Chonofsky, Russell Coleman, Wei Huang, Karen Barber, Nam Nguyen, Stéphane Morin, Fernando Tardio, and Paul Stoop.

Appendix B

Copyright Detail

Mas gusto ko sana na i-license ang libro sa ilalim ng mas less restrictive na CC-BY-SA license. Pero sa kasamaang-palad may ilang unscrupulous organizations na naghahanap at nakakahanap ng freely licensed books, at pagkatapos nag-publish at nagbebenta ng halos walang pagbabagong kopya ng mga libro sa print on demand service tulad ng LuLu o KDP. Ang KDP ay (thankfully) nagdagdag ng policy na binibigay ang preference sa wishes ng actual copyright holder kaysa sa non-copyright holder na nagtatangkang mag-publish ng freely licensed work. Sa kasamaang-palad may maraming print-on-demand services at kakaunti lang ang may well-considered na policy tulad ng KDP.

Sa kasamaang-palad, dinagdag ko ang NC element sa license ng librong ito para bigyan ako ng recourse kung sakaling may mag-tangkang i-clone ang librong ito at ibenta ito commercially. Sa kasamaang-palad, ang pagdagdag ng NC ay naglilimita sa paggamit ng material na ito na gusto kong pahintulutan. Kaya dinagdag ko ang section na ito ng document para ilarawan ang specific situations kung saan binibigay ko ang permission ko in advance na gamitin ang material sa librong ito sa situations na maaaring iturin ng iba bilang commercial.

- Kung magpi-print ka ng limited number ng copies ng lahat o parte ng librong ito para gamitin sa course (e.g., tulad ng coursepack), binigyan ka ng CC-BY license sa mga materials na ito para sa purpose na iyon.
- Kung ikaw ay isang teacher sa university at itranslate mo ang librong ito sa language na iba sa English at magturo gamit ang translated book, maaari kang makipag-ugnayan sa akin at bibigyan kita ng CC-BY-SA license sa mga materials na ito tungkol sa publication ng iyong translation. Sa partikular, papayagan kang magbenta ng resulting translated book commercially.

Kung plano mong i-translate ang libro, maaari kang makipag-ugnayan sa akin para masiguro na mayroon ka ng lahat ng related course materials para ma-translate mo rin sila.

Syempre, welcome ka na makipag-ugnayan sa akin at humingi ng permission kung ang mga clauses na ito ay hindi sapat. Sa lahat ng cases, ang permission na muling gamitin at remix ang material na ito ay ibibigay basta may clear added value o benefit sa mga estudyante o teachers na makukuha bilang resulta ng bagong work.

Charles Severance
www.dr-chuck.com
Ann Arbor, MI, USA
September 9, 2013

Index