1

Introduzione

Obiettivi del capitolo

- Comprendere il significato delle parole "computer" e "programmazione"
- Scrivere ed eseguire il primo programma Python
- Riconoscere gli errori che si manifestano durante la compilazione e l'esecuzione
- Descrivere algoritmi facendo uso di pseudocodice

In questo capitolo apprenderete le basi che vi consentiranno di iniziare a programmare, un'attività che richiede lo studio di un progetto e la predisposizione di un piano per affrontarlo mediante la scelta di strumenti opportuni. Dopo una rapida panoramica sui componenti hardware e software di un computer e, in generale, sulla programmazione, imparerete a scrivere ed eseguire il vostro primo programma Python, imparando anche a individuare e correggere gli errori di programmazione. Infine, vedrete come, durante la progettazione di programmi, si possano descrivere semplici *algoritmi* (cioè descrizioni in più fasi della soluzione di un problema) facendo uso di pseudocodice.

1.1 Computer e programmi

I computer eseguono istruzioni molto elementari a velocità molto elevata.

Molto probabilmente avete già usato un computer, per lavoro o per svago. Molte persone utilizzano i computer per attività quotidiane, come calcolare il saldo di un conto corrente bancario o scrivere un elaborato. I computer sono ottimi per questi lavori, perché possono gestire operazioni ripetitive, come sommare numeri o inserire parole in una pagina, senza annoiarsi o stancarsi.

La flessibilità di un computer è davvero stupefacente: la medesima macchina può calcolare il saldo di un conto corrente, impaginare una relazione e consentirvi di giocare. Al contrario, altre macchine svolgono una gamma di attività più ristretta: un'automobile viaggia e un tostapane tosta il pane. I computer sono in grado di eseguire compiti così diversi perché eseguono programmi diversi, ciascuno dei quali descrive al computer come si possa risolvere un problema ben determinato.

Un programma per computer è una sequenza di istruzioni da eseguire e di decisioni da prendere.

Di per sé, un computer è una macchina che immagazzina dati (numeri, parole, immagini), interagisce con dispositivi (lo schermo, gli altoparlanti, la stampante) ed esegue programmi. I **programmi** descrivono a un computer, in estremo dettaglio, la sequenza di passi che devono essere eseguiti per portare a compimento un'attività o risolvere un problema. In un computer, distinguiamo i componenti **hardware** (la parte fisica di un calcolatore e i suoi dispositivi periferici) da quelli **software**, costituiti dai programmi che vengono eseguiti dall'hardware.

Gli attuali programmi per computer sono talmente sofisticati che è difficile credere che siano composti interamente da operazioni estremamente semplici. Eccone qualche esempio:

- accendi un punto rosso in questa posizione dello schermo;
- somma questi due numeri;
- se questo valore è negativo, prosegui l'esecuzione da quella istruzione.

L'utente del computer percepisce un'interazione fluida soltanto perché i programmi contengono un numero enorme di tali istruzioni elementari, che vengono eseguite ad altissima velocità.

La programmazione è l'arte di progettare e realizzare programmi per i computer.

Chiamiamo **programmazione** l'arte di progettare e realizzare programmi per i computer. Studiando questo libro imparerete a programmare un computer, cioè a fornirgli le istruzioni necessarie a portare a termine compiti specifici.

Scrivere il programma per un sofisticato gioco per computer, con effetti sonori e immagini in movimento, o per un elaboratore di testi (*word processor*) che consenta l'uso di vari tipi di caratteri e l'inserimento di immagini è un compito complesso, che richiede una squadra di molti programmatori altamente specializzati. I vostri primi esercizi di programmazione saranno più elementari, ma i concetti e le competenze che apprenderete in questo libro costituiscono comunque una base importante e non dovrete scoraggiarvi se i vostri primi programmi non potranno competere con il software professionale che siete abituati a utilizzare. Scoprirete che anche semplici attività di programmazione sono molto stimolanti: vedere il computer svolgere con precisione e rapidità un'attività che avrebbe richiesto ore di fatica è un'esperienza assai gratificante, almeno tanto quanto constatare che piccole modifiche a un programma possono portare a miglioramenti immediati: sarete portati a considerare il computer come un'estensione delle vostre capacità cognitive.

1.2 L'anatomia di un computer

Per comprendere il processo di programmazione, è necessario conoscere almeno per grandi linee gli elementi costitutivi di un computer. Pertanto, daremo uno sguardo a un personal computer: gli elaboratori di maggiori dimensioni hanno componenti più veloci, più grandi o più potenti, ma hanno sostanzialmente la stessa struttura.

Nel cuore del computer si trova l'**unità centrale di elaborazione** (CPU, *central processing unit*, in Figura 1), i cui collegamenti interni sono enormemente complicati: nel periodo in cui è stato scritto questo libro, le CPU usate nei personal computer erano costituite da parecchie centinaia di milioni di elementi strutturali, detti *transistor*.

Figura 1
Unità centrale di elaborazione (CPU)

La CPU esegue le istruzioni del programma e ne elabora i dati.

La CPU esegue le istruzioni di controllo del programma e ne elabora i dati, cioè: individua ed esegue le istruzioni del programma; effettua le operazioni aritmetiche, quali addizioni, sottrazioni, moltiplicazioni e divisioni; reperisce dati dalla memoria esterna o dai dispositivi periferici, oppure ve li memorizza.

I dispositivi di memorizzazione comprendono la memoria principale e la memoria secondaria.

Esistono due tipi di memoria. La **memoria principale** (o primaria) è costituita da *chip* di memoria: circuiti elettronici che, quando alimentati da energia elettrica, sono in grado di memorizzare dati. La **memoria secondaria**, che generalmente è un **disco rigido** (*hard disk*, in Figura 2), consente una registrazione dei dati più lenta ma meno costosa, che perdura anche in assenza di alimentazione elettrica. Un disco rigido è formato da piatti rotanti, rivestiti da materiale magnetico, e da testine di lettura/scrittura, in grado di leggere e di modificare il flusso magnetico sui piatti.

Un computer memorizza dati e programmi. Entrambi si trovano nella memoria secondaria e vengono caricati nella memoria principale nel momento in cui inizia l'esecuzione di uno specifico programma, il quale, poi, aggiorna i dati nella memoria principale e li archivia, così modificati, nella memoria secondaria.

Per interagire con un utente umano, un computer ha bisogno di dispositivi periferici: trasmette le informazioni (in uscita, *output*) mediante uno schermo di

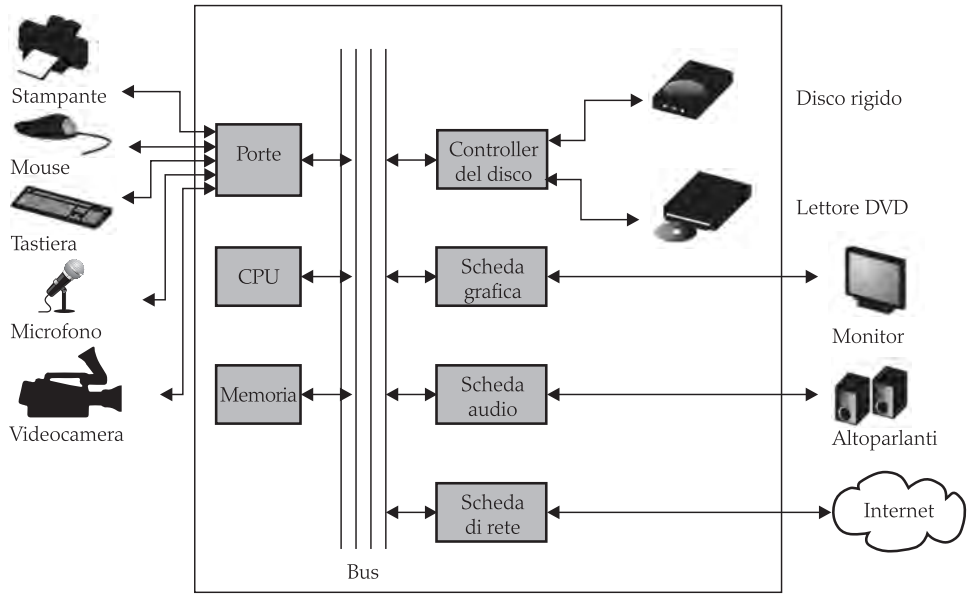
Figura 2
Un disco rigido

visualizzazione, gli altoparlanti e le stampanti. L'utente può fornire informazioni (in ingresso, *input*) al computer tramite una tastiera o un dispositivo di puntamento, quale un mouse.

Alcuni computer sono unità autosufficienti, mentre altri sono connessi tra loro tramite **reti**. Attraverso la connessione alla rete, il computer può leggere dati e programmi da un dispositivo di archiviazione centralizzato oppure inviare dati ad altri computer. Per l'utente di un computer connesso in rete, non sempre può essere semplice distinguere i dati che risiedono sulla propria macchina da quelli che vengono trasmessi attraverso la rete stessa.

La Figura 3 presenta una visione schematica dell'architettura di un personal computer. Le istruzioni dei programmi e i relativi dati (quali testi, numeri, sequenze audio o video) si trovano nel disco rigido, in un disco ottico (come un DVD) o in qualche punto di una rete. Quando si mette in esecuzione un programma, questo viene copiato nella memoria (principale), dove la CPU può leggerlo, un'istruzione dopo l'altra. A seconda delle direttive espresse da tali istruzioni, la CPU legge dati, li modifica e li scrive nuovamente nella memoria principale o secondaria, mentre alcune istruzioni del programma indurranno la CPU a interagire con lo schermo, con la stampante o con l'altoparlante. Poiché queste azioni si ripetono molte volte e a grande velocità, l'utente umano percepirà immagini e suoni. Alcune istruzioni del programma ricevono i comandi dell'utente tramite la tastiera o il mouse: il programma esamina la natura di questi comandi ed esegue conseguentemente le istruzioni appropriate.

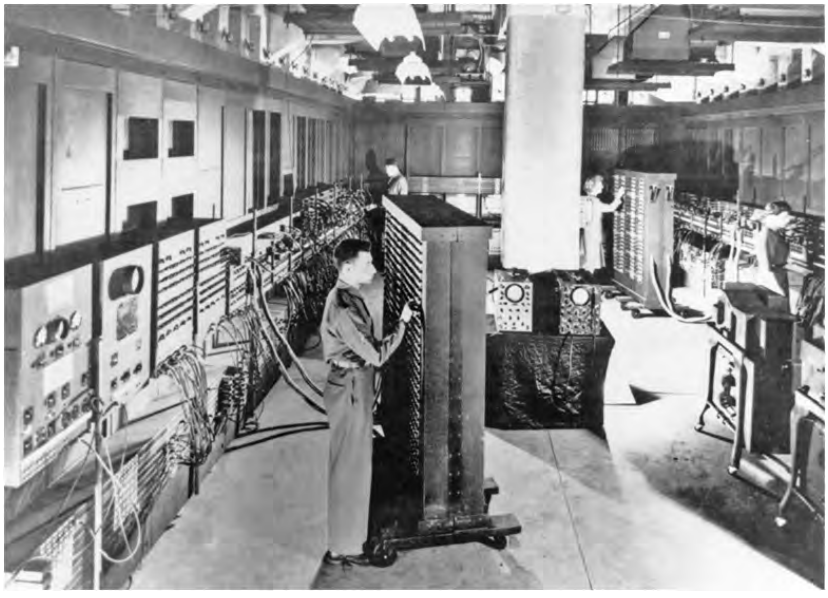
Figura 3
Visione schematica di un personal computer



Computer e società 1.1

I computer sono ovunque

Quando vennero inventati i calcolatori, negli anni Quaranta, uno solo di essi occupava un'intera stanza. La fotografia mostra l'ENIAC (Electronic Numerical Integrator And Computer, integratore numerico e calcolatore elettronico), progettato e costruito nel 1946 presso la University of Pennsylvania: venne usato dall'esercito per calcolare le traiettorie dei proiettili. Oggi, i calcolatori dedicati al funzionamento di motori di ricerca, negozi *online* e *social network* occupano enormi edifici, che prendono il nome di *data center* (centri di elaborazione dei dati). All'estremità opposta troviamo piccoli computer che riempiono la vita attorno a noi: il vostro telefono cellulare contiene un computer, così come ne hanno uno molte carte di credito e abbonamenti per le reti di trasporto pubblico; un'autovettura moderna contiene diversi calcolatori, che ne controllano i freni, il motore, le luci e la radio.



L'ENIAC

L'avvento dell'elaborazione diffusa (*ubiquitous computing*) ha cambiato molti aspetti della nostra vita. In passato, le fabbriche impiegavano personale per portare a termine

Questa tessera di abbonamento contiene un computer

compiti ripetitivi che oggi vengono svolti da robot controllati da computer, sotto la supervisione di poche unità di personale che sa come operare con tali calcolatori. Spesso si fruisce dell'arte (musica, film e libri) tramite un computer e oggi i calcolatori sono quasi sempre coinvolti anche nella produzione di tali opere artistiche. Anche il libro che state leggendo non avrebbe potuto essere scritto senza l'utilizzo di calcolatori.

Conoscere i computer e saperli programmare è diventato essenziale per fare carriera in molti settori. Gli ingegneri progettano automobili controllate da calcolatori e apparecchiature medicali con funzioni salva-vita. Gli informatici sviluppano programmi che aiutano le persone a rimanere in contatto per migliorare le condizioni sociali: ad esempio, molti attivisti hanno usato i *social network* per condividere filmati che mostravano abusi messi in atto da regimi totalitari e repressivi, e queste informazioni hanno innescato cambiamenti nella pubblica opinione. Dal momento che i computer, grandi e piccoli, sono diventati sempre più diffusi nella vita quotidiana, è sempre più importante che ognuno di noi sappia come funzionano e come li si utilizza. Usando questo libro per imparare a programmare un computer, approfondirete in modo sufficiente le vostre conoscenze dei fondamenti dell'informatica e ciò vi renderà migliori anche come cittadini, oltre che, forse, come professionisti del settore dell'informazione.

1.3 Il linguaggio di programmazione Python

Per scrivere un programma per calcolatore, occorre delineare una sequenza di istruzioni che possano essere eseguite dalla sua CPU. Un programma, quindi, è costituito da un gran numero di istruzioni semplici, comprensibili per la CPU, e scrivere tale lungo elenco, un'istruzione dopo l'altra, è estremamente noioso ed è facile fare errori. Proprio per questo motivo, sono stati sviluppati i **linguaggi di programmazione di alto livello**, che consentono ai programmatori di specificare le azioni che devono essere compiute da un programma ad un livello di astrazione sufficientemente elevato. Le istruzioni ad alto livello vengono, poi, tradotte in modo automatico nelle più dettagliate istruzioni che sono necessarie alla CPU.

In questo libro useremo un linguaggio di programmazione di alto livello che si chiama Python ed è stato sviluppato all'inizio degli anni Novanta da Guido van Rossum, con l'obiettivo di eseguire compiti ripetitivi nella gestione di sistemi di elaborazione: l'autore non era soddisfatto dei linguaggi disponibili, che erano stati ottimizzati per agevolare la scrittura di programmi veloci e di grandi dimensioni, perché doveva, invece, progettare programmi piccoli, senza particolari esigenze di velocità di esecuzione. Riteneva, al contrario, importante poter programmare velocemente e modificare i programmi altrettanto rapidamente, qualora questo si rendesse necessario. Di conseguenza, van Rossum progettò un linguaggio che gli consentisse di operare agevolmente con dati complessi e articolati e, a partire da quel progetto iniziale, Python si è evoluto in modo significativo: in questo libro ne usiamo la versione 3. Van Rossum è ancora l'autore principale del linguaggio, ma oggi lo sforzo di sviluppo è distribuito tra molti volontari.

Python è portabile e facile da imparare e da usare.

Python si è diffuso in modo rilevante per applicazioni professionali, scientifiche e accademiche, è veramente molto adatto per imparare a programmare e il suo successo è dovuto a molti fattori. Innanzitutto, ha una sintassi molto più semplice e chiara di quella di altri linguaggi molto diffusi, come Java, C e C++, rendendo così più agevole il suo appren-

dimento. Inoltre, è possibile usare un ambiente interattivo per sperimentare l'esecuzione di brevi programmi scritti in Python, una caratteristica che agevola in modo rilevante la transizione per chi è abituato a programmare in altri linguaggi. Infine, Python è estremamente portabile da un sistema di elaborazione a un altro: il medesimo programma Python verrà eseguito, senza alcuna modifica, su sistemi Windows, UNIX, Linux o Macintosh.

Un pacchetto (o modulo) contiene codice dedicato a risolvere problemi in un dominio specifico.

Oggi molti programmatori scelgono Python per la grande disponibilità di pacchetti: moduli di codice dedicato a risolvere problemi in un dominio specifico. Esistono pacchetti che si occupano dei domini più disparati, come la bioinformatica, l'apprendimento automatico (*machine learning*), la statistica, la visualizzazione dei dati e molti altri. Si tratta di moduli scritti da programmatori esperti e spesso vengono messi a disposizione gratuitamente: usando tali pacchetti, potrete sfruttare nei vostri progetti l'esperienza di chi li ha realizzati. Ad esempio, potreste utilizzare un pacchetto dedicato all'apprendimento automatico per individuare schemi ricorrenti nei vostri dati, per poi presentare i risultati usando un pacchetto di visualizzazione. In questo libro presenteremo alcuni di questi utili pacchetti, all'interno di sezioni facoltative denominate "Strumenti".

1.4 L'ambiente di programmazione

Dedicate un po' di tempo all'apprendimento dell'ambiente di programmazione che dovrete utilizzare.

Molti studenti ritengono che gli strumenti software che si utilizzano durante la programmazione siano molto diversi da quelli a cui sono abituati: è necessario prendere confidenza con l'ambiente di programmazione. Dato che i sistemi operativi dei computer sono molto variegati, in questo testo forniremo solamente una traccia schematica dei passi da seguire: sarebbe opportuno partecipare ad attività di laboratorio oppure chiedere la guida di un amico competente.

Fase 1 Installare l'ambiente di sviluppo per il linguaggio Python.

Probabilmente il vostro docente vi avrà fornito istruzioni per l'installazione dell'ambiente di sviluppo usato nel corso. In caso contrario, potete seguire le istruzioni che trovate all'indirizzo `http://horstmann.com/python4everyone/install.html`.

Fase 2 Mettere in esecuzione l'ambiente di sviluppo Python.

Un editor è un programma che aiuta a creare e modificare un testo (ad esempio, un programma Python).

Per l'esecuzione di questa fase, si riscontrano grandi differenze tra i diversi sistemi operativi. Molti computer mettono a disposizione un **ambiente di sviluppo integrato** (IDE, *integrated development environment*), nel quale scrivere e collaudare i programmi. In altri, occorre per prima cosa mettere in esecuzione un **editor di testo** (*text editor*), cioè un programma che funziona in modo analogo a un *word processor* ("elaboratore di testi"), dove si scrivono le istruzioni in linguaggio Python, per poi aprire una **finestra di terminale**, in cui scrivere i comandi che eseguono il programma. Seguite le istruzioni fornite dal vostro docente o, in mancanza, quelle che trovate all'indirizzo `http://horstmann.com/python4everyone/install.html`.

Fase 3 Scrivere un semplice programma.

Per convenzione consolidata, nell'apprendere un nuovo linguaggio di programmazione si inizia con un programma che visualizza un semplice saluto: "Hello, World!". Seguiamo la tradizione, quindi ecco il programma "Hello, World!" in Python:

```
# Il mio primo programma scritto in Python.
print("Hello, World!")
```

Nel prossimo paragrafo analizzeremo questo programma.

Indipendentemente dall'ambiente di programmazione utilizzato, l'attività inizia con la scrittura delle istruzioni del programma all'interno della finestra di un editor.

Usando i comandi previsti dal vostro ambiente di programmazione, create un nuovo file e chiamatelo `hello.py` (se, oltre al nome del file, l'ambiente richiede di dare un nome al progetto, chiamatelo `hello`). Scrivete le istruzioni copiandole *esattamente* dall'esempio fornito. In alternativa, potete individuare il file nella raccolta di codice Python scaricabile dalla pagina Web dedicata al libro e copiarne il contenuto usando l'editor.

Python distingue tra lettere maiuscole e minuscole, quindi occorre fare attenzione.

Mentre copiate questo programma, fate molta attenzione ai vari simboli che vi compaiono e ricordate che Python *distingue fra lettere maiuscole e lettere minuscole*, quindi dovete inserire le lettere, maiuscole o minuscole, esattamente come appaiono nel testo: non si può scrivere `Print` o `PRINT`. Se non si fa attenzione, si incorre in problemi (si veda la sezione Errori comuni 1.1).

Fase 4 Eseguire il programma.

Le modalità di esecuzione di un programma dipendono, di nuovo, dall'ambiente di programmazione utilizzato: può essere sufficiente un click con il mouse, oppure può essere richiesta la digitazione di comandi. In seguito all'esecuzione del programma (Figure 4 e 5), da qualche parte, sullo schermo, comparirà il messaggio seguente:

```
Hello, World!
```

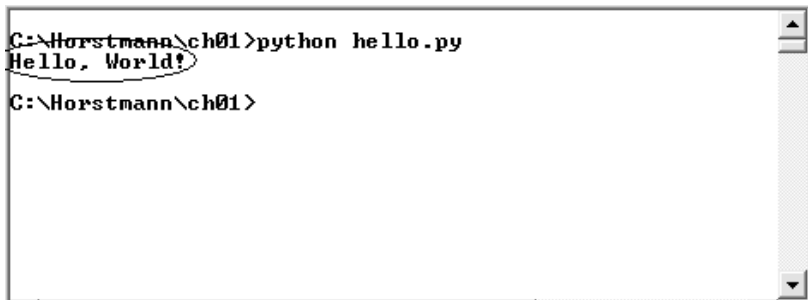


Figura 4
Esecuzione del programma hello.py in una finestra di terminale

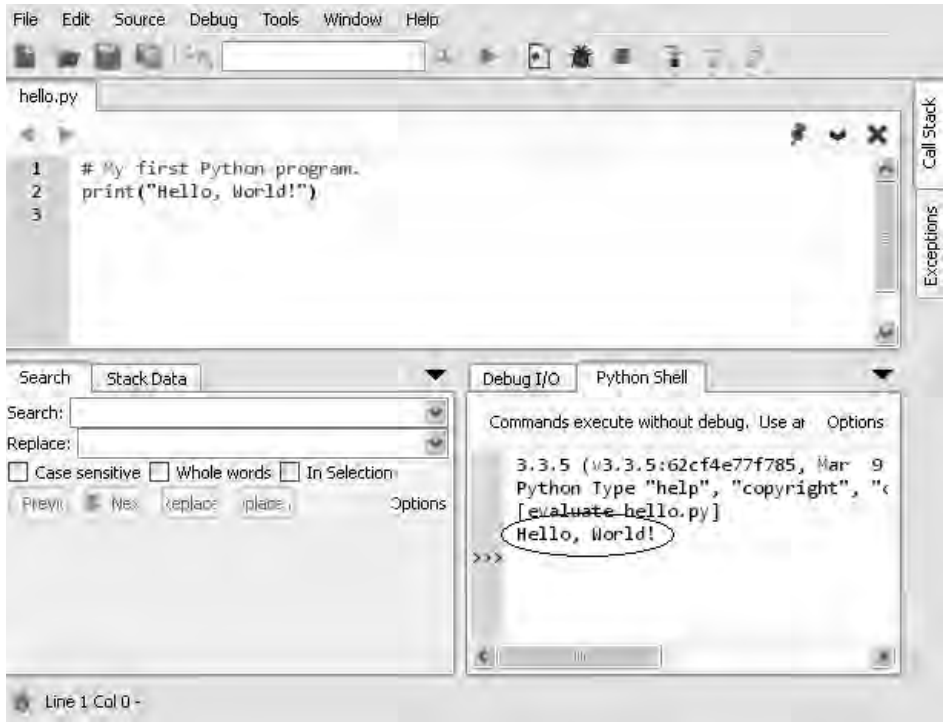
L'interprete Python legge i programmi Python e ne esegue le istruzioni.

Un programma Python viene eseguito usando l'**interprete Python**, che legge il programma e ne esegue le istruzioni (come illustrato in maggiore dettaglio nella sezione Argomenti avanzati 1.1). In alcuni ambienti di programmazione l'interprete Python viene eseguito automaticamente nel momento in cui si seleziona il comando "Run" (o "Esegui"), mentre in altri occorre metterlo in esecuzione in modo esplicito.

Fase 5 Organizzare il lavoro.

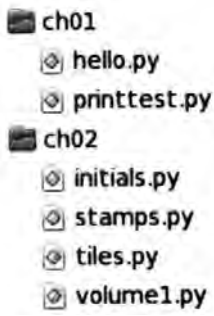
L'attività del programmatore prevede di scrivere programmi, eseguirli e migliorarli. Se volete conservare un programma, magari per presentarlo all'esame, lo dovete archiviare in un **file**, Un programma Python può essere memorizzato in un file avente qualunque nome, a patto che termini con `.py`: infatti, il nostro primo programma è stato memorizzato nel file `hello.py`, oppure avrebbe potuto essere memorizzato con il nome `welcome.py`.

Figura 5
Esecuzione del programma hello.py in un ambiente di sviluppo integrato



I file vengono archiviati all'interno di **cartelle** (*folder* o *directory*). Una cartella può contenere file e/o altre cartelle, le quali, a loro volta, possono contenere altri file e/o cartelle (come si può vedere nella Figura 6). Questa gerarchia può essere piuttosto estesa e non c'è bisogno che ne conosciate tutte le diramazioni, anche se è preferibile che creiate cartelle per organizzare il vostro lavoro. Ad esempio, è bene avere una cartella che contenga le esercitazioni svolte durante il corso di programmazione e, al suo interno, si creerà una cartella per ciascun programma.

Figura 6
Una gerarchia di cartelle



Alcuni ambienti di programmazione posizionano i programmi in una cartella predefinita ("di *default*"), a meno che non se ne specifichi una: in tal caso, dovrete scoprire quale sia.

Assicuratevi di aver capito dove si trovano i vostri file all'interno della gerarchia completa di cartelle: questa informazione sarà essenziale quando dovrete sottoporre a

valutazione i file prodotti, oltreché per predisporne una copia di sicurezza (*backup*, si veda la sezione Suggerimenti per la programmazione 1.2).

Suggerimenti per la programmazione 1.1

Modalità interattiva

Quando si scrive un programma completo, si inseriscono le sue istruzioni in un file e lo si fa eseguire dall'interprete Python. L'interprete, però, è dotato anche di una modalità interattiva, che consente di scrivere le istruzioni una alla volta. Per eseguire l'interprete Python in modalità interattiva all'interno di una finestra di terminale, basta scrivere il comando:

```
python
```

(se nel sistema sono state installate più versioni di Python, si consiglia di usare il comando `python3` per mettere in esecuzione la versione 3 dell'interprete); inoltre, la modalità interattiva può essere attivata anche tramite la maggior parte degli ambienti di sviluppo integrato.

L'interfaccia che consente di lavorare in modalità interattiva è nota come **shell di Python**. Per prima cosa vedrete un messaggio informativo, simile a questo:

```
Python 3.1.2 (r312:79147, Aug 23 2010, 05:17:13)
[GCC 4.4.4 20100630 (Red Hat 4.4.4-10)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

I caratteri >>> che compaiono in basso costituiscono il **prompt** e indicano il punto in cui è possibile scrivere istruzioni in linguaggio Python. Dopo aver scritto un'istruzione e aver premuto il tasto "Invio" (o "Enter"), il codice viene immediatamente eseguito dall'interprete Python. Se, ad esempio, scriviamo:

```
print("Hello, World!")
```

l'interprete risponderà eseguendo la funzione `print` e visualizzando i dati in uscita, seguiti da un nuovo prompt:

```
>>> print("Hello, World!")
Hello, World!
>>>
```

La modalità interattiva si rivela particolarmente utile nella fase iniziale dell'apprendimento del linguaggio, perché consente di fare esperimenti e collaudi di singole istruzioni Python, osservando i risultati prodotti. Si può usare la modalità interattiva anche come semplice calcolatrice, basta scrivere le espressioni matematiche mediante la sintassi di Python:

```
>>> 7035 * 0.15
1055.25
>>>
```

Prendete l'abitudine di utilizzare la modalità interattiva ogni volta che vi accingete a sperimentare nuovi costrutti sintattici del linguaggio.

Suggerimenti per la programmazione 1.2

Adottare una strategia di *backup*

Prima che si verifichi un disastro irrecuperabile, pianificate una strategia di backup del vostro lavoro.

Trascorrerete molte ore scrivendo programmi Python e cercando di migliorarli, ma è facile cancellare accidentalmente un file e a volte i file vanno perduti a causa di un malfunzionamento del computer. Digitare nuovamente il contenuto di un file perduto è frustrante e dispendioso, in termini di tempo, per cui è cruciale saper salvaguardare i propri file e prendere l'abitudine di farlo *prima* che avvenga un disastro. Occorre fare copie di sicurezza (o di *backup*) dei propri file, mettendoli in salvo in una chiavetta di memoria (*memory stick*) o in qualche server della rete Internet, soluzione che sta diventando sempre più diffusa. Ecco alcuni punti da tenere presente:

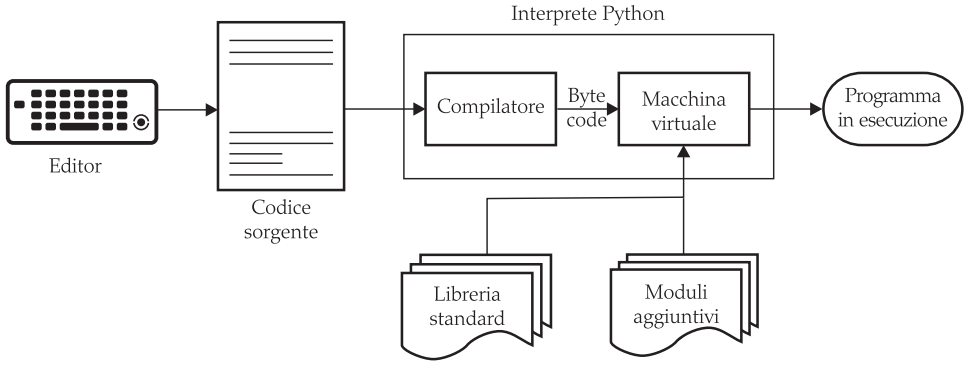
- *Fate backup spesso*. Salvare un file richiede solo pochi secondi e, se doveste perdere molte ore per ricreare un lavoro che avreste potuto salvare facilmente, vi detesterete: è bene salvare il proprio lavoro almeno ogni trenta minuti.
- *Utilizzate i supporti di backup a rotazione*. Usate destinazioni fisicamente diverse per i backup, a rotazione: per prima cosa eseguire il salvataggio su un primo supporto, poi sul secondo, quindi sul terzo, per poi, infine, riutilizzare il primo. In questo modo, avrete sempre i tre salvataggi più recenti: anche se uno fosse difettoso, potrete usare uno dei rimanenti.
- *Fate attenzione alla direzione del backup.* L'azione di backup consiste nel copiare file da un posto a un altro: è importante farlo nella direzione corretta, copiando i file che volete salvaguardare dalla cartella di lavoro alla cartella di backup. Se lo fate nel modo sbagliato, sovrascriverete un file più recente con una versione più vecchia.
- *Controllate i backup di tanto in tanto.* Controllate accuratamente che le vostre copie di backup siano dove pensate. Non c'è nulla di più frustrante della scoperta che, quando se ne ha bisogno, i backup non ci siano.
- *Rilassatevi prima di ripristinare.* Quando perderete un file e avrete bisogno di ripristinarlo dal supporto di backup, probabilmente sarete nervosi: prima di iniziare, fate un respiro profondo e riflettete sul processo di recupero. Non è raro che un utente in preda all'agitazione distrugga il backup, nel tentativo di ripristinare un file danneggiato.

Argomenti avanzati 1.1

L'interprete Python

Quando usate l'interprete Python per eseguire un programma, potete immaginare che legga il programma ed esegua un'istruzione dopo l'altra, ma la realtà non è proprio questa. Dato che spesso un programma viene eseguito più volte, la fase di lettura e interpretazione delle istruzioni, che richiede tempo, viene eseguita una volta sola, da un componente detto **compilatore**. Il compilatore legge il file contenente il **codice sorgente** (*source code*, cioè le istruzioni Python che avete scritto) e traduce le istruzioni in **byte code** ("codice a byte"), un insieme di istruzioni estremamente elementari che possono essere comprese da una **macchina virtuale**, un altro programma che ha un comportamento

Figura 7
Dal codice sorgente al programma in esecuzione



analogo a quello della CPU di un computer. Dopo che il compilatore ha tradotto il programma in istruzioni per la macchina virtuale, queste vengono eseguite da essa ogni volta che lo desideriamo.

Il codice sorgente non contiene tutte le informazioni di cui ha bisogno la macchina virtuale, ad esempio, non contiene l'implementazione della funzione `print`, per cui la macchina virtuale recupera tali funzioni da moduli di libreria, di cui, in generale, non vi occuperete. Ciò nonostante, se volete risolvere problemi in un ambito specifico, come la programmazione grafica, è possibile che dobbiate installare le librerie necessarie: i dettagli di questa procedura dipendono dall'ambiente Python che utilizzate.

Nel vostro *file system* possono essere presenti file contenenti istruzioni per la macchina virtuale: hanno estensione `.pyc` e sono stati generati dal compilatore. Non c'è bisogno che li alleghiate al codice sorgente quando lo inviate al docente per una valutazione, perché sono utili soltanto alla macchina virtuale Python e non sono facilmente comprensibili da un utente umano.

1.5 Analisi del primo programma

In questo paragrafo analizzeremo in dettaglio il primo programma Python, che abbiamo già presentato. Ecco di nuovo il suo codice sorgente:

File ch01/sec04/hello.py

```
1 # Il mio primo programma scritto in Python.
2 print("Hello, World!")
```

Un programma scritto in linguaggio Python contiene una o più righe di istruzioni o **enunciati**, che verranno tradotti ed eseguiti dall'interprete Python. La prima riga

```
# Il mio primo programma scritto in Python.
```

Un commento fornisce informazioni ai programmatori.

è un **commento**. I commenti iniziano con il carattere `#` e non sono enunciati: servono a fornire informazioni ai programmatori e ne parleremo più diffusamente nel Paragrafo 2.1.5.

La seconda riga contiene un enunciato:

```
print("Hello, World!")
```

Una funzione è un insieme di istruzioni che eseguono un compito specifico.

che visualizza (o "stampa", *print*) una riga di testo, in questo caso "Hello, World!". In questo enunciato invochiamo una funzione che si chiama `print` e le passiamo le informazioni che devono essere visualizzate. Una **funzione** è un insieme di istruzioni di programmazione che servono a portare a termine un compito specifico. Non c'è bisogno di realizzare (o "implementare") questa funzione, perché fa parte del linguaggio Python. L'unica cosa che ci interessa sapere è che la funzione esegue il compito per cui è stata progettata: visualizzare un valore.

Una funzione viene invocata specificandone il nome e gli argomenti.

In Python, per usare (o "invocare") una funzione, occorre specificare:

1. il nome della funzione che si vuole utilizzare (in questo caso, `print`);
2. i valori che servono alla funzione per portare a termine il proprio compito (in questo caso, `"Hello, World!"`), che, tecnicamente, vengono detti **argomenti** e vanno racchiusi tra una coppia di parentesi tonde, separando un argomento dall'altro mediante una virgola; il numero di argomenti necessari dipende dalla funzione.

Una sequenza di caratteri racchiusa tra virgolette (singole o doppie), come questa:

```
"Hello, World!"
```

Una stringa è una sequenza di caratteri racchiusa da una coppia di virgolette, singole o doppie.

viene chiamata **stringa**. Il contenuto di una stringa va racchiuso tra virgolette perché sia evidente che si vuole esprimere l'informazione `"Hello, World!"` in modo letterale. C'è un motivo ben preciso per l'esistenza di questo requisito. Immaginate di voler visualizzare la parola *print*: racchiudendola tra virgolette, si rende chiaro che `"print"` significa "la sequenza di caratteri `p r i n t`", non il nome della funzione `print`. La regola è molto semplice e prevede di racchiudere tutte le stringhe all'interno di una coppia di virgolette, semplici o doppie.

Si possono anche visualizzare valori numerici. Ad esempio, l'enunciato:

```
print(3 + 4)
```

Sintassi 1.1 Enunciato print

Sintassi

```
print()
print(valore1, valore2, …, valoren)
```

Esempio

Tutti gli argomenti sono facoltativi: se non viene fornito alcun argomento, viene visualizzata una riga vuota.

```
print("The answer is", 6 + 7, "!")
```

I valori che verranno visualizzati, uno dopo l'altro e separati da uno spazio.

valuta l'espressione 3 + 4 e ne visualizza il risultato, il numero 7. È possibile passare più argomenti a una funzione e questo esempio:

```
print("The answer is", 6 * 7)
```

visualizza `The answer is 42`. La funzione `print` visualizzerà tutti i valori che le vengono passati, uno dopo l'altro, nello stesso ordine in cui sono stati scritti, separandoli con uno spazio. Inoltre, dopo aver visualizzato i propri argomenti, la funzione `print` va a capo. Ad esempio, gli enunciati:

```
print("Hello")
print("World!")
```

visualizzano due righe di testo:

```
Hello
World!
```

Se non si fornisce alcun argomento alla funzione `print`, viene semplicemente iniziata una nuova riga di testo (cioè la visualizzazione "va a capo"), in modo analogo a quanto succede in un editor di testo quando si preme il tasto "Enter" o "Invio".

Ad esempio, gli enunciati:

```
print("Hello")
print()
print("World")
```

visualizzano tre righe di testo, la seconda delle quali è una riga vuota:

```
Hello

World
```

In un programma Python, tutti gli enunciati devono iniziare nella stessa colonna. Ad esempio, il programma seguente non è valido, perché il livello dei rientri verso destra ("indentazione") non è coerente:

```
print("Hello")
   print("World")
```

Ecco, infine, un programma che illustra in modo abbastanza articolato l'utilizzo della funzione `print`.

File ch01/sec05/printtest.py

```
1  ##
2  #  Programma che illustra il comportamento della funzione print.
3  #
4
5  # Visualizza 7.
6  print(3 + 4)
7
8  # Visualizza "Hello World!" su due righe.
```

```
 9  print("Hello")
10  print("World!")
11
12  # Visualizza più valori usando una singola invocazione di print.
13  print("My favorite numbers are", 3 + 4, "and", 3 + 10)
14
15  # Visualizza tre righe di testo, la seconda delle quali vuota.
16  print("Goodbye")
17  print()
18  print("Hope to see you again")
```

Esecuzione del programma

```
7
Hello
World!
My favorite numbers are 7 and 13
Goodbye

Hope to see you again
```

1.6 Errori

Facciamo qualche esperimento con il programma `hello.py`. Vediamo che cosa succede con errori come questo:

```
print("Hello, World!)
```

(si notino le virgolette mancanti al termine del messaggio di saluto). Quando tentiamo di eseguire il programma, l'interprete si blocca e visualizza il messaggio seguente:

```
File "hello.py", line 2
   print("Hello, World!)
                       ^
SyntaxError: EOL while scanning string literal
```

Un errore di compilazione è una violazione delle regole del linguaggio di programmazione identificata durante la traduzione in formato eseguibile.

Si tratta di un **errore (nella fase) di compilazione** (come detto nella sezione Argomenti avanzati 1.1, chiamiamo *compilazione* il processo di trasformazione delle istruzioni Python in una forma eseguibile): in base alle regole del linguaggio c'è qualcosa di sbagliato e il compilatore lo rileva prima che il programma venga effettivamente eseguito. Per questo motivo, spesso gli errori identificati in fase di compilazione sono anche detti **errori di sintassi**. Quando il compilatore individua errori di questo tipo, non viene creato alcun programma eseguibile: dovete rimediare all'errore e tentare di nuovo l'esecuzione. Di fatto, l'interprete è piuttosto esigente e non è raro passare attraverso numerosi cicli di correzione degli errori di sintassi, prima di poter eseguire un programma per la prima volta. In questo caso la correzione è semplice: basta aggiungere le virgolette al termine della stringa.

Sfortunatamente l'interprete non è particolarmente brillante e spesso non aiuta in alcun modo nell'identificazione dell'errore di sintassi presente nel codice. Supponiamo, ad esempio, di dimenticare le virgolette che racchiudono una stringa, tanto quelle iniziali quanto quelle finali, scrivendo:

```
print(Hello, World!)
```

L'errore viene segnalato in questo modo:

```
File "hello.py", line 2
   print(Hello, World!)
                      ^
SyntaxError: invalid syntax
```

Tocca a voi capire che è necessario racchiudere la stringa tra virgolette.

Alcuni errori possono essere identificati soltanto durante l'esecuzione del programma. Supponiamo, ad esempio, che il programma contenga l'enunciato seguente:

```
print(1 / 0)
```

Questo enunciato non viola le regole sintattiche del linguaggio Python e l'esecuzione del programma inizia, ma, nel momento in cui avviene la divisione per zero, il programma si interrompe e viene visualizzato il seguente messaggio d'errore:

```
Traceback (most recent call last):
   File "hello.py", line 3, in <module>
ZeroDivisionError: int division or modulo by zero
```

Si verifica un'eccezione quando un'istruzione è sintatticamente corretta, ma non può essere eseguita.

Si tratta di una **eccezione** (*exception*). Diversamente da un errore di sintassi, che viene segnalato nel momento in cui l'interprete analizza il codice del programma, un'eccezione si manifesta durante l'esecuzione del programma stesso: un'eccezione è un **errore (nella fase) di esecuzione**.

Ecco un diverso tipo di errore di esecuzione:

```
print("Hello, Word!")
```

Il programma è sintatticamente corretto e viene eseguito senza alcuna eccezione, ma non produce il risultato previsto: invece di visualizzare il messaggio "Hello, World!", visualizza "Hello, Word!", scrivendo "Word" al posto di "World".

Un errore di esecuzione è un errore che avviene durante l'esecuzione del programma, che produce risultati inattesi.

A volte si parla di **errore logico** anziché di errore di esecuzione: dopo tutto, quando il programma si comporta in modo errato, questo avviene in conseguenza di un errore nella logica del programma. In un programma ben scritto non saranno presenti divisioni per zero, né verranno visualizzati messaggi errati.

Durante lo sviluppo dei programmi, gli errori sono inevitabili. Quando un programma supera le poche righe, occorre una concentrazione sovrumana per digitarlo correttamente, senza incorrere in alcuna svista. Vi sorprenderete a compiere errori di ortografia, a dimenticare virgolette e a cercare di eseguire un'operazione non lecita più spesso di quanto vi piacerebbe ammettere, ma, per fortuna, il compilatore scoverà questi errori per voi, e li potrete correggere.

Gli errori logici sono più fastidiosi. È più difficile scoprirli e correggerli, perché l'interprete non li segnala: è responsabilità dell'autore del programma collaudarlo e scoprire eventuali errori di esecuzione.

Errori comuni 1.1

Errori di ortografia

Se sbagliate accidentalmente l'ortografia di una parola, possono succedere strane cose e non sempre è facile capire dai messaggi di errore che cosa è andato storto. Ecco un buon esempio che illustra come banali errori di ortografia possano essere fastidiosi:

```
Print("Hello, World!")
print("How are you?")
```

Il primo enunciato invoca la funzione `Print`, che è diversa dalla funzione `print`, perché `Print` inizia con una lettera maiuscola e il linguaggio Python distingue tra lettere maiuscole e minuscole. Le lettere maiuscole e minuscole sono considerate come lettere tra loro completamente diverse e, per l'interprete Python, `Print` non assomiglia a `print` più di `pint`. Naturalmente, il messaggio `name 'Print' is not defined` (cioè "il nome `'Print'` non è definito") dovrebbe essere un valido indizio per aiutarvi nella ricerca dell'errore.

Se un messaggio di errore sembra indicare che l'interprete Python sia fuori strada, è il caso di controllare l'ortografia e le lettere maiuscole e minuscole.

1.7 Soluzione di problemi: progettazione di algoritmi

Presto imparerete ad eseguire calcoli e a prendere decisioni mediante la programmazione in Python. Prima, però, di prendere in esame, nei capitoli successivi, i dettagli tecnici che consentono di eseguire calcoli, consideriamo come si possano descrivere i passi che si rendono necessari per trovare la soluzione di un problema assegnato.

Può darsi che vi sia capitato di vedere annunci pubblicitari che vi esortano a pagare per un servizio computerizzato che trova "l'anima gemella". Provate a pensare a come questo possa funzionare: dovete compilare un questionario e inviarlo, così come faranno altre persone; i dati così raccolti vengono, poi, elaborati al computer da un programma. Ha senso ipotizzare che il computer sia in grado di individuare le persone che meglio corrispondono al vostro profilo?

Immaginate che sia vostro fratello più giovane, invece del computer, ad avere sulla propria scrivania tutti i questionari compilati: che istruzioni gli dareste? Non potete pensare di dirgli, semplicemente "Trova la persona più carina che ama pattinare e navigare in Internet", perché non esiste un metodo oggettivo per valutare la bellezza ed è molto probabile che l'opinione di vostro fratello (così come quella di un computer che analizzi fotografie digitalizzate) sia diversa dalla vostra. Se non siete in grado di fornire a qualcuno istruzioni scritte che consentano di risolvere un problema, non c'è modo che il computer possa trovare magicamente una soluzione: il computer può fare soltanto ciò che gli dite di fare, anche se è in grado di farlo più velocemente e senza annoiarsi o affaticarsi. Proprio per questo motivo, non c'è alcuna garanzia che un servizio computerizzato che trova "l'anima gemella" possa fornire il risultato migliore.

Esaminiamo, invece, il seguente problema relativo a un investimento finanziario:

> Depositate 10000 dollari in un conto bancario che fornisce il 5 per cento di interesse annuo. Quanti anni occorrono perché il saldo del conto arrivi al doppio dell'importo iniziale?

Sareste in grado di risolvere il problema facendo i calcoli manualmente? Dovreste senza dubbio riuscirci, in questo modo:

anno	interessi	saldo
0		10 000
1	10 000.00 × 0.05 = 500.00	10 000.00 × 1.05 = 10 500.00
2	10 500.00 × 0.05 = 525.00	10 500.00 × 1.05 = 11 025.00
3	11 025.00 × 0.05 = 551.25	11 025.00 × 1.05 = 11 576.25
4	11 576.25 × 0.05 = 578.81	11 576.25 × 1.05 = 12 155.06

Basta proseguire finché il saldo non è almeno pari a 20000 dollari: a quel punto, la risposta è l'ultimo numero che avete scritto nella colonna relativa all'anno.

Lo svolgimento di calcoli di questo tipo è, ovviamente, assai noioso, sia per voi sia per il vostro fratello più giovane, mentre i calcolatori si comportano ottimamente nell'esecuzione veloce di calcoli ripetitivi, senza fare errori. Ciò che è importante per il calcolatore è una descrizione dei passi necessari per trovare la soluzione del problema: ogni passo deve essere espresso in modo chiaro e non ambiguo, senza che ci sia bisogno di "indovinare". Ecco una descrizione di questo tipo:

Iniziare ponendo anno uguale a 0 e saldo uguale a 10000.

anno	interessi	saldo
0		10 000

Ripetere i passi seguenti finché il saldo è minore di 20000.
  Aggiungere 1 al valore dell'anno.
  Calcolare gli interessi come "saldo per 0.05" (cioè 5 per cento).
  Aggiungere gli interessi al saldo.

anno	interessi	saldo
0		10 000
1	500.00	10 500
⋮		⋮
14	942.82	19 799.32
⑮	989.96	20 789.28

Riportare il valore finale dell'anno come risposta.

Lo pseudocodice è una descrizione informale di una sequenza di passi che portano alla soluzione di un problema.

Come è evidente, questi passi non sono ancora descritti in un linguaggio comprensibile al calcolatore, ma imparerete ben presto a formularli in Python. Questa descrizione informale è detta **pseudocodice**.

Non esistono requisiti precisi e stringenti per lo pseudocodice, perché non è destinato alla lettura da parte di programmi, bensì di persone. Ecco i tipi di enunciati in pseudocodice che useremo nel libro:

- Per descrivere come viene impostato o modificato un valore:

```
costo totale = prezzo di acquisto × costo operativo
Moltiplicare il valore del saldo per 1.05.
Eliminare dalla parola il primo e l'ultimo carattere.
```

- Per descrivere decisioni e ripetizioni:

```
Se costo totale 1 < costo totale 2
Finché il saldo è minore di 20000
Per ogni immagine presente nella sequenza
```

  Per indicare quali enunciati siano oggetto di selezione o di ripetizione, si usa l'indentazione, cioè il rientro del testo verso destra:

```
Per ogni automobile
   costo operativo = 10 × costo annuale del carburante
   costo totale = prezzo di acquisto + costo operativo
```

  L'indentazione usata in questo esempio indica che entrambi gli enunciati devono essere eseguiti per ogni automobile.

- Per descrivere i risultati prodotti:

```
Scegliere automobile1.
Riportare il valore finale dell'anno come risposta.
```

Non è importante usare esattamente le stesse parole usate qui. Ciò che importa è che lo pseudocodice descriva una sequenza di passi:

- che non sia ambigua;
- che sia eseguibile;
- che termini in un tempo finito.

Un passo di una sequenza è *non ambiguo* quando contiene istruzioni precise in merito a cosa si debba fare e specifica con certezza quale sia il passo successivo, senza lasciare spazio alle opinioni personali e all'inventiva. Un passo è, poi, *eseguibile* quando può essere effettivamente portato a termine. Se in un passo si chiedesse di usare il tasso di interesse reale dei prossimi anni, invece di un tasso di interesse annuo prefissato e pari al 5 percento, esso non sarebbe eseguibile, perché non c'è modo di conoscere i tassi di interesse futuri. Infine, una sequenza di passi *termina in un tempo finito* se, prima o poi, ha certamente termine. Nel nostro esempio, bisogna fare un piccolo ragionamento per dimostrare che la sequenza di passi non verrà ripetuta all'infinito: ad ogni passo il saldo aumenta di almeno 500 dollari, per cui raggiungerà certamente il valore di 20000 dollari.

Un algoritmo che risolve un problema è una sequenza di passi non ambigua, eseguibile e che termina in un tempo finito.

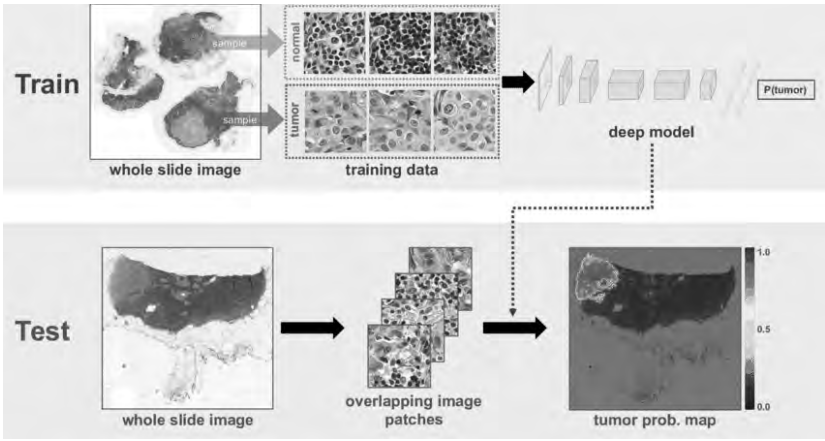
Una sequenza di passi che non sia ambigua, che sia eseguibile e che termini in un tempo finito prende il nome di **algoritmo**. Dato che abbiamo individuato un algoritmo che risolve il nostro problema finanziario, possiamo risolverlo anche programmando un calcolatore: l'esistenza di un algoritmo è prerequisito essenziale per la programmazione. Prima di iniziare a programmare, dovete scoprire e descrivere un algoritmo adeguato al compito che volete risolvere, seguendo le fasi delineate nella Figura 8.

Figura 8
Procedimento per lo sviluppo di un programma

Computer e società 1.2

I dati sono ovunque

Il genere umano è solito analizzare i dati relativi all'ambiente circostante, per rispondere a domande quali "Dove atterrerà la palla?" oppure "Quali articoli saranno maggiormente apprezzati dal mercato?"
Aumentando le possibilità di raccogliere dati, siamo sempre alla ricerca di nuovi metodi che ci aiutino ad elaborarli e a prendere decisioni conseguenti.
Nel diciottesimo e diciannovesimo secolo, scienziati e ingegneri hanno imparato a comprendere il mondo fisico attraverso modelli matematici. Pur con la modesta capacità di calcolo che era disponibile, i loro strumenti preferiti furono di tipo computazionale: fare calcoli. Il ventesimo secolo, poi, ha portato in auge i metodi statistici e, con l'avvento degli elaboratori digitali, si è diffusa la possibilità di costruire modelli di sistemi che non potevano essere gestiti in modo efficace dalle regole del calcolo matematico.
Oggi, la facilità con cui si possono raccogliere dati e la grande potenza di calcolo disponibile promuovono metodi innovativi per l'analisi dei dati, spesso chiamati genericamente "scienza dei dati" (*data science*). Un aspetto chiave della scienza dei dati è quello che viene chiamato *data mining*: letteralmente, la scoperta di schemi (*pattern*) ricorrenti e relazioni all'interno di enormi quantità di dati, come se si cercasse l'oro scavando in una miniera. Usando strumenti matematici si possono identificare gruppi di dati tra loro correlati (*cluster*) e metodi statistici consentono di "classificarli", raggruppandoli in diverse categorie. Ad esempio, un'azienda può identificare diversi gruppi di clienti sulla base dei loro comportamenti d'acquisto esibiti in passato, per poi destinare a ciascun gruppo suggerimenti e promozioni specifiche. Le azioni di *data mining* sono utili anche per individuare comportamenti anomali, che possono essere caratteristici di una frode o di un pericolo imminente.
Un altro aspetto molto promettente della data science è l'apprendimento automatico (*machine learning*), un settore che progetta sistemi che tentano di riprodurre i meccanismi di elaborazione del cervello umano. Tali sistemi possono essere addestrati (con attività di *training*) mediante la lettura di alcuni *pattern* già inseriti in categorie: dopo tale fase, saranno in grado di riconoscere *pattern* simili. Ad esempio, è possibile costruire una "rete neurale" (*neural net*) e addestrarla con migliaia di foto di gatti



L'apprendimento automatico (*machine learning*) può aiutare i medici a individuare tumori maligni.

e di cani; in seguito, la rete neurale sarà in grado di distinguere la foto di un gatto dalla foto di un cane con grande affidabilità. A un osservatore superficiale questo potrebbe non sembrare particolarmente sorprendente, ma pensate a come si potrebbe scrivere un programma che risolva tale problema, data una matrice di *pixel* colorati. Come descrivereste un algoritmo che sia in grado di distinguere un gatto da un cane, usando soltanto frasi come "se il *pixel* che si trova nella posizione (x, y) è di colore z, esegui…"

La scienza dei dati (*data science*) è artefice di molti sviluppi in vari settori, come le automobili a guida autonoma e la diagnosi medica assistita dal calcolatore. L'applicazione dei metodi della scienza dei dati è utile a professionisti di varie discipline e, per raggiungere tali obiettivi, è sempre richiesta una significativa attività di programmazione: Python è un linguaggio eccellente in tale ambito, perché esiste una notevole quantità di pacchetti di elevata qualità che mettono a disposizione strumenti tipici della *data science*. Ne vedrete alcuni semplici esempi nelle sezioni "Strumenti" di questo libro. Inoltre, va sottolineato che Python consente uno stile di programmazione "esplorativo": si possono rapidamente sperimentare idee e algoritmi in un ambiente interattivo, fino a quando non vengono identificati *pattern* nei dati. Leggendo questo libro imparerete gli elementi fondamentali del linguaggio Python, che vi serviranno per avere successo nel settore della *data science*.

Consigli pratici 1.1

Descrizione di un algoritmo mediante pseudocodice

Questa è la prima sezione di "Consigli pratici" (*How to*) che trovate in questo libro: si tratta di sezioni che descrivono in dettaglio il procedimento che occorre seguire per portare a termine alcuni compiti specifici e importanti nell'attività di sviluppo di programmi per il calcolatore.

Prima di iniziare a scrivere un programma in Python, dovete sviluppare un algoritmo (cioè un metodo che consenta di giungere alla soluzione di uno specifico problema) e descriverlo mediante pseudocodice: un sequenza di passi ben precisi, espressi in linguaggio naturale.

Descrizione del problema. Dovete scegliere, per l'acquisto, tra due automobili, una delle quali è più efficiente dell'altra in merito al consumo di carburante, ma è più costosa. Vi sono noti il prezzo e l'efficienza (in miglia per gallone, mpg) di entrambe le vetture. Pianificate di usare l'automobile per dieci anni, percorrendo 15 000 miglia all'anno, con un prezzo del carburante di $ 4 al gallone. Pagate l'acquisto in contanti e non prendete in considerazione costi finanziari. Qual è l'acquisto migliore?

Fase 1 Determinare i dati disponibili (*ingressi*) e i risultati da produrre (*uscite*).

Nel nostro esempio, abbiamo questi valori di "ingresso":

- prezzo 1 e efficienza 1, rispettivamente il prezzo di acquisto e l'efficienza (in mpg) della prima automobile
- prezzo 2 e efficienza 2, rispettivamente il prezzo di acquisto e l'efficienza (in mpg) della seconda automobile

Vogliamo semplicemente sapere quale automobile sia meglio acquistare: questo è il risultato cercato.

Fase 2 Scomporre il problema in compiti più semplici.

Dobbiamo sapere il costo totale di ciascuna automobile: facciamo i calcoli separatamente, per la prima e per la seconda. Noto il costo totale di ciascuna, possiamo decidere quale sia la migliore.

> Il costo totale, per ciascuna auto, è prezzo + costo di esercizio.

Nell'ipotesi che, per dieci anni, ci sia un utilizzo costante, con un prezzo costante del carburante, il costo (totale) di esercizio dipende dal costo di esercizio annuale.

> Il costo di esercizio è 10 × spesa annuale per carburante.
> La spesa annuale per carburante è prezzo al gallone × consumo annuale.

Il consumo annuale è pari a miglia percorse all'anno / efficienza. Ad esempio, guidando per 15000 miglia con un'efficienza di 15 miglia/gallone, l'automobile consuma 1000 galloni.

Fase 3 Descrivere ciascun sotto-problema mediante pseudocodice.

Nella descrizione precedente, elencate i vari passi in modo che ogni valore intermedio venga calcolato prima che serva per altri calcoli. Ad esempio, il calcolo:

```
costo totale = prezzo + costo di esercizio
```

deve essere eseguito dopo aver calcolato il costo di esercizio.

Ecco, infine, l'algoritmo completo che consente di decidere quale automobile costituisca l'acquisto migliore.

```
Per ogni automobile, calcolare il costo totale, in questo modo:
   consumo annuale = miglia percorse all'anno / efficienza
   spesa annuale per carburante = prezzo al gallone × consumo annuale
   costo di esercizio = 10 × spesa annuale per carburante
   costo totale = prezzo + costo di esercizio
Se costo totale 1 < costo totale 2
   Scegliere automobile 1
Altrimenti
   Scegliere automobile 2
```

Fase 4 Collaudare lo pseudocodice in casi specifici.

Useremo, come esempio, questi valori:

Automobile 1: $ 25 000, 50 miglia/gallone
Automobile 2: $ 20 000, 30 miglia/gallone

Ecco i calcoli che forniscono il costo totale della prima automobile

consumo annuale = miglia percorse all'anno / efficienza = 15 000 / 50 = 300
spesa annuale per carburante = prezzo al gallone × consumo annuale = 4 × 300 = 1200
costo di esercizio = 10 × spesa annuale per carburante = 10 × 1200 = 12000
costo totale = prezzo + costo di esercizio = 25000 + 12000 = 37000

Analogamente, il costo totale per la seconda automobile risulta essere pari a $ 40 000, per cui l'algoritmo produce come decisione la scelta della prima vettura.

Esempi completi 1.1

Sviluppare un algoritmo per posare piastrelle su un pavimento

Dovete ricoprire il pavimento rettangolare di un bagno con piastrelle, alternativamente bianche e nere, quadrate, con lato di 4 pollici. Le dimensioni del pavimento, misurate in pollici, sono entrambe multiple di 4.

Fase 1 Determinare i dati disponibili (*ingressi*) e i risultati da produrre (*uscite*).

I dati disponibili sono le dimensioni del pavimento (lunghezza × larghezza), misurate in pollici. Il risultato da produrre è un pavimento ricoperto di piastrelle.

Fase 2 Scomporre il problema in compiti più semplici.

Un sotto-problema che si evidenzia in modo naturale è la posa di una riga di piastrelle. Se siete in grado di risolvere questo problema, allora potete portare a termine il compito iniziale posando una riga accanto all'altra, iniziando da un muro, finché non raggiungete il muro opposto.

Come si posa una riga? Iniziate con una piastrella vicino a un muro. Se è bianca, posatele accanto una piastrella nera; se, invece, è nera, posatele accanto una piastrella bianca. Continuate finché non raggiungete il muro opposto. La riga conterrà un numero di piastrelle pari a larghezza / 4.

Fase 3 Descrivere ciascun sotto-problema mediante pseudocodice.

Scrivendo lo pseudocodice, vogliamo essere ancora più precisi nell'enunciare dove vadano posizionate esattamente le piastrelle.

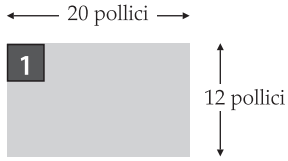
Posare una piastrella nera nell'angolo nord-occidentale.
Finché il pavimento non è completamente ricoperto, ripetere questi passi:
  Ripetere questo passo ((larghezza / 4) – 1) volte:
    Se la piastrella posata precedentemente era bianca,

```
        prenderne una nera, altrimenti prenderne una bianca.
      Posare la piastrella a est di quella posata precedentemente.
    Identificare la piastrella iniziale della riga appena posata.
    Se a sud c'è spazio, posare là una piastrella di colore diverso.
```

Fase 4 Collaudare lo pseudocodice in casi specifici.

Immaginate di voler pavimentare una superficie che misura 20 × 12 pollici.

Il primo passo consiste nella posa di una piastrella nera nell'angolo nord-occidentale.



Poi, posate alternativamente quattro piastrelle, fino al raggiungimento del muro a est (larghezza / 4 – 1 = 20 / 4 – 1 = 4).

Verso sud c'è ancora spazio. Individuate la piastrella che si trova all'inizio della riga appena completata: è nera, quindi posate una piastrella bianca a sud di essa.

Completate la riga.

Verso sud c'è ancora spazio. Individuate la piastrella che si trova all'inizio della riga appena completata: è bianca, quindi posate una piastrella nera a sud di essa.

Completate la riga.

A questo punto il pavimento è completamente ricoperto: avete finito.

Riepilogo del capitolo

Definizione di "programma per calcolatore" e di "programmazione"

- I computer eseguono istruzioni molto elementari a velocità molto elevata.
- Un programma per computer è una sequenza di istruzioni da eseguire e di decisioni da prendere.
- La programmazione è l'arte di progettare e realizzare programmi per i computer.

Descrizione dei componenti di un computer

- La CPU esegue le istruzioni del programma e ne elabora i dati.
- I dispositivi di memorizzazione comprendono la memoria principale e la memoria secondaria.

Descrizione dei vantaggi del linguaggio Python

- Python è portabile e facile da imparare e da usare.
- Un pacchetto (o modulo) contiene codice dedicato a risolvere problemi in un dominio specifico.

L'ambiente di programmazione Python

- Dedicate un po' di tempo all'apprendimento dell'ambiente di programmazione che dovrete utilizzare.
- Un editor è un programma che aiuta a creare e modificare un testo (ad esempio, un programma Python).
- Python distingue tra lettere maiuscole e minuscole, quindi occorre fare attenzione.
- L'interprete Python legge i programmi Python e ne esegue le istruzioni.
- Prima che si verifichi un disastro irrecuperabile, pianificate una strategia di backup del vostro lavoro.

Descrizione dei blocchi costituivi di un semplice programma

- Un commento fornisce informazioni ai programmatori.
- Una funzione è un insieme di istruzioni che eseguono un compito specifico.
- Una funzione viene invocata specificandone il nome e gli argomenti.
- Una stringa è una sequenza di caratteri racchiusa da una coppia di virgolette, singole o doppie.

Errori di compilazione ed errori di esecuzione

- Un errore di compilazione è una violazione delle regole del linguaggio di programmazione identificata durante la traduzione in formato eseguibile.
- Si verifica un'eccezione quando un'istruzione è sintatticamente corretta, ma non può essere eseguita.
- Un errore di esecuzione è un errore che avviene durante l'esecuzione del programma, che produce risultati inattesi.

Scrittura di semplici algoritmi mediante pseudocodice

- Lo pseudocodice è una descrizione informale di una sequenza di passi che portano alla soluzione di un problema.
- Un algoritmo che risolve un problema è una sequenza di passi non ambigua, eseguibile e che termina in un tempo finito.

Esercizi di ripasso

* **R1.1.** Spiegate la differenza che intercorre fra usare un programma al calcolatore e programmare un computer.

* **R1.2.** Quali parti di un computer possono memorizzare il codice di un programma? E quali possono memorizzare i dati degli utenti?

* **R1.3.** Quali parti di un computer hanno il compito di fornire informazioni agli utenti? E quali possono ricevere informazioni dagli utenti?

*** **R1.4.** Un tostapane è un dispositivo che svolge un'unica funzione, mentre un computer può essere programmato per portare a termini compiti diversi. Il vostro telefono cellulare è un dispositivo analogo a un tostapane o, piuttosto, è programmabile come un computer? La risposta dipende dal modello di telefono di cui disponete.

* **R1.5.** Quali linguaggi di programmazione sono stati menzionati in questo capitolo? Quando sono stati inventati? Da chi? Fate ricerche in Internet.

** **R1.6.** Nel vostro computer personale, oppure in quello del laboratorio, trovate la posizione esatta (cioè il nome della cartella) in cui si trova:

a. Il file di esempio `hello.py`, che avete scritto con l'editor.
b. L'interprete Python, `python` oppure `python.exe`.

** **R1.7.** Cosa visualizza questo programma?

```
print("39 + 3")
print(39 + 3)
```

** **R1.8.** Cosa visualizza questo programma? Fate attenzione agli spazi.

```
print("Hello", "World", "!")
```

** **R1.9.** Quale errore di compilazione è presente in questo programma?

```
print("Hello", "World!)
```

** **R1.10.** Scrivete tre versioni del programma `hello.py`, con errori di compilazione diversi. Scrivetene, poi, una versione con un errore di esecuzione.

* **R1.11.** In che modo si scoprono gli errori di compilazione? Come si scoprono gli errori di esecuzione?

** **R1.12.** Scrivete un algoritmo per rispondere a questa domanda: un conto bancario contiene inizialmente $ 10 000; gli interessi vengono calcolati mensilmente, al tasso mensile dello 0.5 per cento; ogni mese vengono prelevati $ 500 per pagare le spese universitarie; dopo quanti anni il conto è vuoto?

★★★ **R1.13.** Considerate di nuovo l'esercizio precedente e supponete che i valori che vi figurano ($ 10 000, 0.5 per cento, $ 500) siano a discrezione dell'utente del programma. Esistono valori che impediscono all'algoritmo che avete sviluppato di terminare? In caso affermativo, modificate l'algoritmo in modo da avere la certezza che termini sempre.

★★★ **R1.14.** Per fare il preventivo del costo di tinteggiatura di una casa, l'imbianchino deve conoscere l'area della sua superficie esterna. Sviluppate un algoritmo che calcoli tale valore, avendo come dati di ingresso l'altezza della casa e le sue due dimensioni orizzontali (larghezza e lunghezza), oltre al numero di finestre e porte e alle relative dimensioni (nell'ipotesi che tutte le finestre abbiano le stesse dimensioni, al pari di tutte le porte).

★★ **R1.15.** Volete decidere se andare al lavoro in automobile o in treno. Conoscete la distanza tra casa vostra e il luogo di lavoro, oltre all'efficienza della vostra automobile nel consumo di carburante (in miglia percorse per gallone consumato). Sapete anche il prezzo del biglietto ferroviario di sola andata. Ipotizzate che il carburante costi 4 dollari al gallone e che il costo di manutenzione dell'automobile sia pari a 5 centesimi di dollaro per ogni miglio percorso. Scrivete un algoritmo per decidere la modalità più economica per fare il pendolare.

★★ **R1.16.** Volete calcolare la percentuale di utilizzo della vostra automobile per uso personale e, separatamente, per recarvi al lavoro. Conoscete la distanza tra casa vostra e il vostro luogo di lavoro e, per un certo periodo, avete registrato il valore iniziale e finale riportato dal contachilometri, oltre al numero di giorni lavorativi. Scrivete un algoritmo per risolvere questo problema.

★ **R1.17.** Nel problema descritto nei Consigli pratici 1.1, si sono fatte ipotesi sul prezzo del carburante e sull'utilizzo annuale dell'automobile. In linea teorica, sarebbe meglio sapere quale vettura sia la migliore senza dover fare tali ipotesi. Perché un programma eseguito al calcolatore non è in grado di risolvere tale problema?

★★★ **R1.18.** Il valore di π può essere calcolato con la seguente formula:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Scrivete un algoritmo che calcoli π. Dato che la formula è una sommatoria infinita e un algoritmo deve terminare dopo aver eseguito un numero finito di passi, interrompete il calcolo quando sono state determinate sei cifre significative del risultato.

★★ **R1.19.** Immaginate di dover dare al vostro fratello minore l'incarico di eseguire il backup del vostro lavoro. Scrivete una sequenza dettagliata di istruzioni da seguire per portare a termine il compito assegnato. Spiegate la frequenza con cui questo va fatto e quali file debbano essere copiati, da quale cartella a quale altra cartella o dispositivo. Spiegate anche come si verifichi che il backup sia stato effettuato correttamente.

★ **R1.20 (economia).** Supponete di andare in un ristorante di lusso insieme ad alcuni amici e che, al momento di pagare il conto, vogliate dividere in parti uguali, compresa la mancia del 15%. Descrivete, mediante pseudocodice, un algoritmo che calcoli la somma dovuta da ciascuno. Il programma deve visualizzare l'importo del conto, la mancia, il costo totale e la somma dovuta da ciascuno; inoltre, deve visualizzare la somma dovuta da ciascuno per pagare il conto e, separatamente, per la mancia.

Esercizi di programmazione

★ **P1.1.** Scrivete un programma che visualizzi un saluto che vi piace, probabilmente in una lingua diversa dall'inglese.

★★ **P1.2.** Scrivete un programma che visualizzi la somma dei primi 10 numeri interi positivi: 1 + 2 + … + 10.

★★ **P1.3.** Scrivete un programma che visualizzi il prodotto dei primi 10 numeri interi positivi: 1 × 2 × … × 10 (in Python, usate l'asterisco per indicare l'operazione di moltiplicazione).

★★ **P1.4.** Scrivete un programma che visualizzi il saldo di un conto bancario dopo il primo, secondo e terzo anno. Il conto ha un saldo iniziale di 1000 dollari e vi vengono accreditati interessi annuali al 5%.

★ **P1.5.** Scrivete un programma che visualizzi sullo schermo il vostro nome all'interno di un rettangolo, come nell'esempio seguente:

```
+------+
¦ Dave ¦
+------+
```

Fate quanto possibile per comporre i lati del rettangolo con i caratteri ¦ - +.

★★★ **P1.6.** Scrivete un programma che scriva il vostro nome con lettere molto grandi, in questo modo:

```
*   *    **    ****    ****    *   *
*   *   *  *   *   *   *   *   *   *
*****  *    *  ****    ****     * *
*   *  ******  *   *   *   *     *
*   *  *    *  *    *  *    *    *
```

★★ **P1.7.** Scrivete un programma che, usando caratteri, visualizzi un viso, possibilmente migliore di questo:

```
  /////
 ¦ o o ¦
(¦  ^  ¦)
 ¦ \_/ ¦
  -----
```

★★ **P1.8.** Scrivete un programma che visualizzi un'imitazione di un quadro di Piet Mondrian (se non conoscete questo artista, fate una ricerca in Internet). Usate sequenze di caratteri come `@@@` o `:::` per indicare colori diversi, e usate i caratteri - e ¦ per comporre le linee.

★★ **P1.9.** Scrivete un programma che visualizzi una casa identica a questa:

```
   +
  + +
 +   +
+-----+
¦ .-. ¦
¦ ¦ ¦ ¦
+-+-+-+
```

★★★ **P1.10.** Scrivete un programma che visualizzi un animale mentre pronuncia un saluto, simile a questo (ma diverso):

```
 /\_/\     -----
( ' ' )  / Hello \
(  -  ) <  Junior ¦
 ¦ ¦ ¦   \ Coder!/
(__¦__)    -----
```

* **P1.11.** Scrivete un programma che visualizzi tre stringhe (ad esempio i nomi dei vostri migliori amici o i vostri film preferiti) su tre righe consecutive.

* **P1.12.** Scrivete un programma che visualizzi la poesia che preferite (se non ne avete una, cercate "Emily Dickinson" in Internet).

** **P1.13.** Scrivete un programma che, usando i caratteri `*` e `=`, visualizzi la bandiera degli Stati Uniti d'America.

* **P1.14 (economia).** Scrivete un programma che visualizzi, sotto forma di elenco in due colonne, le date dei compleanni dei vostri amici. Nella prima colonna devono comparire i nomi degli amici e nella seconda la data del compleanno corrispondente.

* **P1.15 (economia).** Negli Stati Uniti non ci sono tasse federali sugli acquisti, per cui ogni singolo stato può imporre le proprie tasse. Fate una ricerca in Internet per scoprire il livello di tassazione sugli acquisti in cinque stati di vostra scelta, poi scrivete un programma che visualizzi una tabella come questa:

```
Sales Tax Rates
------------------
Alaska: 0%
Hawaii: 4%
...
```

* **P1.16 (economia).** Nel moderno mercato del lavoro, la capacità di parlare più lingue è un'abilità decisamente apprezzata ed è basilare imparare i saluti. Scrivete un programma che visualizzi un elenco, su due colonne, con frasi di saluto: nella prima colonna scrivete frasi in inglese (ad esempio, "Good morning", "It is a pleasure to meet you", "Please call me tomorrow", "Have a nice day!"), mentre nella seconda scrivetene la traduzione in italiano.