

|                         |             |
|-------------------------|-------------|
| <b>Prefazione</b> ..... | <b>xxix</b> |
|-------------------------|-------------|

---

## **Parte I. Per cominciare**

|   |           |
|---|-----------|
| <b>1. Domande e risposte su Python</b> .....                      | <b>3</b>  |
| Perché usare Python?  | 3         |
| Qualità del software  | 4         |
| Produttività di sviluppo  | 5         |
| Python è un “linguaggio di scripting”?                            | 5         |
| Bene, ma quali sono gli svantaggi?                                | 6         |
| Chi usa Python oggi?  | 7         |
| Cosa si può fare con Python?                                      | 9         |
| Programmazione di sistema   | 9         |
| GUI   | 9         |
| Scripting Internet  | 10        |
| Integrazione di componenti  | 10        |
| Programmazione di database  | 10        |
| Prototipazione rapida   | 11        |
| Programmazione numerica e scientifica                             | 11        |
| Giochi, immagini, porte seriali, XML, robot e altro               | 11        |
| Qual è il supporto di cui gode Python?                            | 12        |
| Quali sono i punti di forza di Python dal punto di vista tecnico? | 12        |
| È orientato agli oggetti  | 12        |
| È open source   | 13        |
| È portabile   | 13        |
| È potente   | 14        |
| È interoperabile  | 15        |
| È facile da usare   | 16        |
| È facile da imparare  | 16        |
| Il suo nome deriva dai Monty Python                               | 16        |
| Quali sono le differenze tra Python e gli altri linguaggi?        | 17        |
| Riassunto   | 19        |
| Domande   | 19        |
| Risposte  | 19        |
| <b>2. Come Python esegue i programmi</b> .....                    | <b>21</b> |
| Introduzione all'interprete Python                                | 21        |
| Esecuzione dei programmi  | 22        |

|  |           |
|--|-----------|
| La prospettiva del programmatore                         | 22        |
| La prospettiva di Python                                 | 24        |
| Modelli d'esecuzione alternativi                         | 26        |
| Implementazioni Python alternative                       | 27        |
| Strumenti per l'ottimizzazione del modello di esecuzione | 28        |
| Binari congelati   | 29        |
| Altre opzioni d'esecuzione                               | 30        |
| Possibilità future?                                      | 30        |
| Riassunto  | 31        |
| Domande  | 31        |
| Risposte   | 32        |
| <b>3. Come voi eseguite i programmi Python .....</b>     | <b>33</b> |
| Il prompt interattivo                                    | 33        |
| Eseguire il codice interattivamente                      | 34        |
| Perché il prompt interattivo?                            | 35        |
| Uso del prompt interattivo                               | 37        |
| Linea di comando e file                                  | 38        |
| Un primo script  | 39        |
| Come eseguire i file dalla riga di comando               | 40        |
| Uso della riga di comando e dei file                     | 41        |
| Script eseguibili Unix (#!)                              | 42        |
| Click sulle icone  | 44        |
| Click sotto Windows                                      | 44        |
| Il trucco input  | 44        |
| Altre limitazioni dei click sulle icone                  | 47        |
| Importazione e reload dei moduli                         | 47        |
| Moduli e attributi                                       | 49        |
| <b>import</b> e <b>reload</b> : note d'uso               | 52        |
| Usare <b>exec</b> per eseguire i file modulo             | 52        |
| L'interfaccia utente di IDLE                             | 53        |
| Uso di base di IDLE                                      | 54        |
| Consigli   | 55        |
| Strumenti avanzati di IDLE                               | 57        |
| Altri ambienti di sviluppo grafici                       | 57        |
| Altre opzioni di esecuzione                              | 59        |
| Chiamate inglobate                                       | 59        |
| Eseguibili contenenti binari congelati                   | 60        |
| Opzioni di esecuzione degli editor di testo              | 60        |
| Ulteriori opzioni di esecuzione                          | 60        |
| Possibilità future?                                      | 60        |
| Quali opzioni usare?                                     | 61        |
| Riassunto  | 62        |
| Domande  | 62        |
| Risposte   | 63        |
| Esercizi per la Parte I                                  | 64        |

---

## Parte II. Tipi e operazioni

|  |           |
|--|-----------|
| <b>4. Introduzione ai tipi di oggetti Python</b> ..... | <b>69</b> |
| Perché usare i tipi precostituiti?                     | 69        |
| I tipi dati di base di Python                          | 70        |
| Numeri   | 72        |
| Stringhe   | 73        |
| Operazioni sulle sequenze                              | 74        |
| Immutabilità   | 76        |
| Metodi specifici dei tipi                              | 76        |
| Ottenerne aiuto  | 77        |
| Altri modi per elaborare le stringhe                   | 78        |
| Ricerca di corrispondenze                              | 79        |
| Liste  | 80        |
| Operazioni sulle sequenze                              | 80        |
| Operazioni specifiche per le liste                     | 80        |
| Verifica degli estremi                                 | 81        |
| Annidamento  | 81        |
| Espressioni di mappatura                               | 82        |
| Dizionari  | 83        |
| Operazioni sulle mappature                             | 84        |
| Annidamento rivisitato                                 | 84        |
| Ordinamento delle chiavi: cicli <b>for</b>             | 86        |
| Iterazioni e ottimizzazione                            | 87        |
| Chiavi mancanti e test <b>if</b>                       | 88        |
| Le tuple   | 89        |
| Perché esistono le tuple?                              | 90        |
| File   | 90        |
| Altri strumenti simili ai file                         | 92        |
| Altri tipi di oggetti base                             | 92        |
| Come scrivere codice poco flessibile                   | 93        |
| Classi definite dall'utente                            | 94        |
| Tutto il resto   | 95        |
| Riassunto  | 95        |
| Domande  | 96        |
| Risposte   | 96        |
| <b>5. Numeri</b> .....                                 | <b>99</b> |
| Le basi dei tipi numerici di Python                    | 99        |
| Letterali numerici                                     | 100       |
| Strumenti numerici precostituiti                       | 101       |
| Operatori per le espressioni                           | 102       |
| Esempi d'uso dei tipi numerici                         | 106       |
| Variabili ed espressioni semplici                      | 106       |
| Formati di visualizzazione numerica                    | 108       |
| Confronti: normali e accodati                          | 109       |

|   |            |
|---|------------|
| Divisione: classica, floor e vera                                 | 110        |
| Precisione degli interi   | 113        |
| Numeri complessi  | 114        |
| Notazione esadecimale, ottale e binaria                           | 114        |
| Operazioni a livello di bit                                       | 116        |
| Altri strumenti numerici predefiniti                              | 117        |
| Altri tipi numerici   | 119        |
| Il tipo decimale  | 119        |
| Il tipo frazione  | 121        |
| Insiemi   | 124        |
| I Booleani  | 130        |
| Estensioni numeriche  | 131        |
| Riassunto   | 131        |
| Domande   | 131        |
| Risposte  | 132        |
| <b>6. Intermezzo sulla tipizzazione dinamica</b>                  | <b>133</b> |
| Il motivo della mancanza delle dichiarazioni di tipo              | 133        |
| Variabili, oggetti e referenze                                    | 133        |
| I tipi sono associati agli oggetti, non alle variabili            | 135        |
| Il garbage collection agisce sugli oggetti                        | 136        |
| Referenze condivise   | 137        |
| Referenze condivise e modifiche sul posto                         | 139        |
| Referenze condivise e uguaglianze                                 | 140        |
| La tipizzazione dinamica è ovunque!                               | 142        |
| Riassunto   | 142        |
| Domande   | 143        |
| Risposte  | 143        |
| <b>7. Stringhe</b>  | <b>145</b> |
| Letterali stringa   | 147        |
| Le stringhe tra singoli e doppi apici hanno lo stesso significato | 147        |
| Le sequenze di escape rappresentano byte speciali                 | 148        |
| Le stringhe raw rendono inutili gli escape                        | 150        |
| Gli apici tripli codificano blocchi di stringhe multi-riga        | 152        |
| Esempi d'uso delle stringhe                                       | 153        |
| Operazioni di base  | 153        |
| Indicizzazione e sezionamento                                     | 154        |
| Strumenti di conversione per le stringhe                          | 158        |
| Modifica delle stringhe   | 160        |
| I metodi di stringa   | 161        |
| Esempio: modifica delle stringhe                                  | 162        |
| Esempio: parsing del testo  | 164        |
| Altri esempi  | 165        |
| Il modulo <code>string</code> originale (eliminato da Python 3.0) | 166        |
| Espressioni di formattazione                                      | 167        |
| Espressioni di formattazione avanzate                             | 169        |

|  |            |
|--|------------|
| Espressioni di formattazione tramite dizionari                   | 170        |
| Formattazione tramite chiamate di metodo                         | 171        |
| Le basi  | 171        |
| Chiavi, attributi e indici                                       | 172        |
| Formattazioni speciali   | 173        |
| Confronto con le espressioni %                                   | 175        |
| Perché il nuovo metodo di formattazione?                         | 177        |
| Categorie generali di tipi                                       | 179        |
| I tipi condividono le operazioni caratteristiche delle categorie | 180        |
| I tipi mutabili possono essere modificati sul posto              | 180        |
| Riassunto  | 181        |
| Domande  | 181        |
| Risposte   | 181        |
| <b>8. Liste e dizionari</b> .....                                | <b>183</b> |
| Liste  | 183        |
| Esempi di liste  | 185        |
| Operazioni di base   | 185        |
| Iterazioni ed espressioni di mappatura                           | 186        |
| Indirizzamento, sezionamento e matrici                           | 187        |
| Modifica sul posto   | 188        |
| Dizionari  | 192        |
| Esempi di dizionari  | 194        |
| Operazioni di base   | 194        |
| Modifica sul posto   | 195        |
| Altri metodi per i dizionari                                     | 196        |
| Una tabella di linguaggi   | 197        |
| Note d'uso per i dizionari                                       | 198        |
| Altri modi per creare dizionari                                  | 201        |
| Le novità sui dizionari in Python 3.0                            | 202        |
| Riassunto  | 207        |
| Domande  | 208        |
| Risposte   | 208        |
| <b>9. Tuple, file e tutto il resto</b> .....                     | <b>209</b> |
| Le tuple   | 209        |
| Esempi d'uso delle tuple   | 210        |
| Perché liste e tuple?  | 213        |
| I file   | 213        |
| Apertura dei file  | 214        |
| Come usare gli oggetti di tipo file                              | 214        |
| Esempi d'uso   | 216        |
| Altri strumenti per i file                                       | 222        |
| Rivisitazione delle categorie generali di tipi                   | 223        |
| Flessibilità   | 224        |
| Referenze e copie  | 225        |
| Confronti, uguaglianza e verità                                  | 227        |

|   |     |
|---|-----|
| I confronti tra dizionari in Python 3.0                   | 229 |
| Il significato di vero e falso in Python                  | 229 |
| La gerarchia dei tipi Python                              | 231 |
| Gli oggetti <b>type</b>                                   | 231 |
| Altri tipi Python   | 233 |
| Trappole d'uso per i tipi base                            | 233 |
| Gli assegnamenti creano referenze, non copie              | 234 |
| Le ripetizioni aggiungono nuovi livelli di annidamento    | 234 |
| Attenzione alle strutture dati cicliche                   | 235 |
| I tipi immutabili non possono essere modificati sul posto | 235 |
| Riassunto   | 236 |
| Domande   | 236 |
| Risposte  | 236 |
| Esercizi relativi alla Parte II                           | 237 |

---

## Parte III. Istruzioni e sintassi

|   |            |
|---|------------|
| <b>10. Introduzione alle istruzioni Python</b> .....      | <b>243</b> |
| Rivisitazione della struttura dei programmi Python        | 243        |
| Le istruzioni di Python                                   | 243        |
| Tutta la sintassi in due <b>if</b>                        | 245        |
| Quello che Python aggiunge                                | 246        |
| Quello che Python elimina                                 | 246        |
| Perché la spaziatura orizzontale?                         | 247        |
| Casi speciali   | 250        |
| Esempio: cicli interattivi                                | 251        |
| Un semplice ciclo interattivo                             | 252        |
| Uso di valori inseriti dall'utente per effettuare calcoli | 253        |
| Gestione degli errori tramite test sugli input            | 253        |
| Gestione degli errori tramite le istruzioni <b>try</b>    | 254        |
| Tre livelli di annidamento                                | 255        |
| Riassunto   | 256        |
| Domande   | 256        |
| Risposte  | 256        |
| <b>11. Assegnamenti, espressioni e <b>print</b></b> ..... | <b>259</b> |
| Istruzioni di assegnamento                                | 259        |
| La sintassi delle istruzioni di assegnamento              | 260        |
| Assegnamenti di sequenza                                  | 261        |
| Spacchettamento esteso di sequenze in Python 3.0          | 264        |
| Assegnamenti multipli                                     | 267        |
| Assegnamenti estesi                                       | 268        |
| Regole per i nomi di variabile                            | 271        |
| Espressioni come istruzioni                               | 274        |
| Espressioni come istruzioni e modifiche sul posto         | 275        |
| Istruzioni <b>print</b>                                   | 276        |

|   |            |
|---|------------|
| La funzione <code>print</code> di Python 3.0  | 276        |
| L'istruzione <code>print</code> di Python 2.6   | 278        |
| Redirezione dello stream di output  | 279        |
| Print indipendenti dalla versione di Python   | 283        |
| Riassunto   | 285        |
| Domande   | 285        |
| Risposte  | 285        |
| <b>12. Test <code>if</code> e regole sintattiche</b> .....                                      | <b>287</b> |
| Le istruzioni <code>if</code>   | 287        |
| Formato generale  | 287        |
| Esempi di base  | 288        |
| Blocchi con opzioni multiple  | 288        |
| Le regole di sintassi Python  | 290        |
| Delimitatori di blocco: regole di indentazione  | 291        |
| Delimitatori di istruzioni: righe e continuazioni di righe                                      | 293        |
| Alcuni casi speciali  | 293        |
| Test di verità  | 295        |
| L'espressione ternaria <code>if/else</code>   | 296        |
| Riassunto   | 299        |
| Domande   | 299        |
| Risposte  | 299        |
| <b>13. Cicli <code>while</code> e <code>for</code></b> .....                                    | <b>301</b> |
| I cicli <code>while</code>  | 301        |
| Formato generale  | 301        |
| Esempi  | 302        |
| <code>break</code> , <code>continue</code> , <code>pass</code> e l' <code>else</code> dei cicli | 302        |
| Formato generale  | 303        |
| <code>pass</code>   | 303        |
| <code>continue</code>   | 304        |
| <code>break</code>  | 305        |
| L' <code>else</code> dei cicli  | 305        |
| I cicli <code>for</code>  | 307        |
| Formato generale  | 308        |
| Esempi  | 308        |
| Tecniche di codifica dei cicli  | 314        |
| Cicli contatore: <code>while</code> e <code>range</code>  | 314        |
| Iterazioni parziali: <code>range</code> e sezionamenti  | 316        |
| Modificare le liste con <code>range</code>  | 316        |
| Attraversamenti paralleli: <code>zip</code> e <code>map</code>                                  | 317        |
| Generare sia gli indici che gli elementi: <code>enumerate</code>                                | 320        |
| Riassunto   | 321        |
| Domande   | 321        |
| Risposte  | 322        |

|  |            |
|--|------------|
| <b>14. Iterazione e mappature di lista, prima parte</b> .....              | <b>323</b> |
| Iteratori: introduzione  | 323        |
| Il protocollo d'iterazione: iteratori di file                              | 324        |
| Iterazione manuale: <code>iter</code> e <code>next</code>                  | 326        |
| Altri iteratori precostituiti  | 328        |
| Mappature di lista: introduzione   | 330        |
| Uso di base delle espressioni di mappatura                                 | 330        |
| Uso delle mappature di lista nell'accesso ai file                          | 331        |
| Sintassi estesa per le espressioni di mappatura di lista                   | 332        |
| Altri contesti d'iterazione  | 333        |
| I nuovi iterabili di Python 3.0  | 337        |
| L'iteratore <code>range</code>   | 337        |
| Gli iteratori di <code>map</code> , <code>zip</code> e <code>filter</code> | 338        |
| Iteratori multipli e singoli   | 339        |
| Gli iteratori delle viste dei dizionari                                    | 340        |
| Altri argomenti sugli iteratori  | 342        |
| Riassunto  | 342        |
| Domande  | 342        |
| Risposte   | 342        |
| <br>   |            |
| <b>15. Documentazione del codice</b> .....                                 | <b>345</b> |
| Le fonti della documentazione del codice Python                            | 345        |
| I commenti <code>#</code>  | 346        |
| La funzione <code>dir</code>   | 346        |
| Le docstring: <code>__doc__</code>   | 347        |
| PyDoc: la funzione <code>help</code>                                       | 349        |
| PyDoc: i report HTML   | 352        |
| I manuali della distribuzione standard                                     | 355        |
| Risorse web  | 355        |
| Libri  | 356        |
| Trappole comuni di codifica  | 356        |
| Riassunto  | 357        |
| Domande  | 358        |
| Risposte   | 358        |
| Esercizi relativi alla Parte III   | 358        |

---

## Parte IV. Funzioni

|  |            |
|--|------------|
| <b>16. Concetti base sulle funzioni</b> .....                        | <b>363</b> |
| Perché usare le funzioni?  | 364        |
| Codifica delle funzioni  | 364        |
| L'istruzione <code>def</code>  | 366        |
| Le <code>def</code> vengono valutate durante l'esecuzione del codice | 366        |
| Un primo esempio: definizione e chiamata                             | 367        |
| Definizione  | 367        |
| Chiamata   | 368        |



|  |            |
|--|------------|
| Il polimorfismo in Python                          | 368        |
| Un secondo esempio: intersezione di sequenze       | 369        |
| Definizione  | 370        |
| Chiamata   | 370        |
| Rivisitazione del polimorfismo                     | 371        |
| Variabili locali                                   | 372        |
| Riassunto  | 372        |
| Domande  | 372        |
| Risposte   | 373        |
| <b>17. Scopi</b> .....                             | <b>375</b> |
| Le basi sugli scopi di Python                      | 375        |
| Le regole sugli scopi                              | 376        |
| Risoluzione dei nomi: la regola LEGB               | 378        |
| Un esempio sugli scopi                             | 379        |
| Lo scopo precostituito                             | 380        |
| L'istruzione <code>global</code>                   | 382        |
| Minimizzare l'uso di nomi globali                  | 383        |
| Minimizzare le modifiche cross-file                | 384        |
| Altri modi per accedere ai nomi globali            | 385        |
| Scopi e funzioni annidate                          | 386        |
| Dettagli sugli scopi annidati                      | 387        |
| Esempi di scopi annidati                           | 387        |
| L'istruzione <code>nonlocal</code>                 | 392        |
| Le basi di <code>nonlocal</code>                   | 393        |
| Esempi d'uso di <code>nonlocal</code>              | 394        |
| Perché <code>nonlocal</code> ?                     | 396        |
| Riassunto  | 399        |
| Domande  | 400        |
| Risposte   | 401        |
| <b>18. Gli argomenti delle funzioni</b> .....      | <b>403</b> |
| Le basi  | 403        |
| Argomenti e referenze condivise                    | 404        |
| Evitare la modifica degli argomenti mutabili       | 406        |
| Simulare parametri di output                       | 407        |
| Modalità speciali di accoppiamento degli argomenti | 408        |
| Le basi  | 408        |
| Sintassi di accoppiamento                          | 409        |
| I dettagli   | 410        |
| Esempi di argomenti keyword e di default           | 411        |
| Argomenti arbitrari                                | 413        |
| Gli argomenti solo keyword di Python 3.0           | 417        |
| Una funzione per calcolare minimi                  | 420        |
| Crediti  | 420        |
| Se volete il massimo dei voti                      | 421        |
| Funzioni generali per gli insiemi                  | 422        |

|   |            |
|---|------------|
| Emulazione della funzione <code>print</code> di Python 3.0                          | 423        |
| Uso degli argomenti solo keyword  | 424        |
| Riassunto   | 426        |
| Domande   | 426        |
| Risposte  | 427        |
| <b>19. Argomenti avanzati sulle funzioni</b>  | <b>429</b> |
| Principi di progettazione delle funzioni  | 429        |
| Funzioni ricorsive  | 431        |
| Somme ricorsive   | 431        |
| Codifiche alternative   | 432        |
| Ricorsività e istruzioni d'iterazione   | 433        |
| Gestione di strutture dati arbitrarie   | 433        |
| Le funzioni come oggetti: attributi e annotazioni                                   | 434        |
| Chiamate indirette  | 434        |
| Introspezione delle funzioni  | 435        |
| Attributi delle funzioni  | 436        |
| Le annotazioni di funzioni di Python 3.0  | 437        |
| Funzioni anonime: le lambda   | 438        |
| Concetti di base sulle lambda   | 439        |
| Perché usare le lambda?   | 440        |
| Come (non) offuscare il codice Python   | 442        |
| Lambda annidate e scopi   | 443        |
| Mappare funzioni a sequenze: <code>map</code>                                       | 444        |
| Programmazione funzionale: <code>filter</code> e <code>reduce</code>                | 445        |
| Riassunto   | 447        |
| Domande   | 447        |
| Risposte  | 447        |
| <b>20. Iterazione e mappature di lista, seconda parte</b>                           | <b>449</b> |
| Espressioni di mappatura e strumenti funzionali                                     | 449        |
| Mappature di liste e <code>map</code>   | 449        |
| Test e cicli annidati: <code>filter</code>  | 450        |
| Espressioni da mappatura e matrici  | 452        |
| Come decifrare le espressioni di mappatura  | 453        |
| Iteratori e funzioni generatrici  | 455        |
| Funzioni generatrici: <code>yield</code> e <code>return</code>                      | 455        |
| Generatori come espressioni: iteratori e mappature                                  | 459        |
| Funzioni generatrici ed espressioni di generazione                                  | 461        |
| Generatori come oggetti a iterazione singola  | 462        |
| Come emulare <code>zip</code> e <code>map</code> tramite gli strumenti d'iterazione | 463        |
| Generazione di valori nei tipi precostituiti e nelle classi                         | 467        |
| Riassunto delle sintassi di mappatura   | 468        |
| Mappature di insiemi e di dizionari   | 469        |
| Sintassi estesa per le mappature di set e di dizionario                             | 469        |
| Cicli e velocità d'esecuzione   | 470        |
| Il modulo <code>time</code>   | 471        |

|  |     |
|--|-----|
| Uno script per misurare il tempo d'esecuzione del codice | 471 |
| Risultati  | 472 |
| Alternative al modulo di timing                          | 473 |
| Altri suggerimenti                                       | 477 |
| Trappole d'uso sulle funzioni                            | 477 |
| I nomi locali sono rilevati staticamente                 | 478 |
| Argomenti di default e oggetti mutabili                  | 479 |
| Funzioni senza <b>return</b>                             | 481 |
| Variabili di ciclo degli scopi esterni                   | 481 |
| Riassunto  | 481 |
| Domande  | 482 |
| Risposte   | 482 |
| Esercizi relativi alla Parte IV                          | 483 |

---

## Parte V. Moduli

|   |            |
|---|------------|
| <b>21. Moduli: il quadro generale</b> .....       | <b>489</b> |
| Perché usare i moduli?                            | 489        |
| L'architettura dei programmi Python               | 490        |
| Come strutturare un programma                     | 490        |
| Importazioni e attributi                          | 491        |
| I moduli della libreria standard                  | 492        |
| Come funzionano le importazioni                   | 493        |
| 1. Ricerca del file del modulo                    | 493        |
| 2. Compilazione del modulo (forse)                | 494        |
| 3. Esecuzione del modulo                          | 494        |
| Il percorso di ricerca dei moduli                 | 495        |
| Come configurare il percorso di ricerca           | 497        |
| Altre varianti del percorso di ricerca dei moduli | 497        |
| La lista <b>sys.path</b>                          | 498        |
| Selezione dei file di modulo                      | 498        |
| Concetti avanzati sulla selezione dei moduli      | 499        |
| Riassunto   | 500        |
| Domande   | 500        |
| Risposte  | 501        |
| <b>22. Codifica di base dei moduli</b> .....      | <b>503</b> |
| Creazione dei moduli                              | 503        |
| Uso dei moduli                                    | 504        |
| L'istruzione <b>import</b>                        | 504        |
| L'istruzione <b>from</b>                          | 504        |
| L'istruzione <b>from *</b>                        | 505        |
| Le importazioni vengono effettuate una sola volta | 505        |
| <b>import</b> e <b>from</b> sono assegnamenti     | 506        |
| Modifiche dei nomi in file diversi                | 507        |
| Equivalenza di <b>import</b> e <b>from</b>        | 507        |

|   |            |
|---|------------|
| Trappole potenziali della istruzione <code>from</code>                  | 508        |
| I moduli come spazi di nomi   | 509        |
| I file generano degli spazi di nomi                                     | 509        |
| Qualificazione dei nomi degli attributi                                 | 511        |
| Importazioni e scopi  | 512        |
| Spazi di nomi annidati  | 513        |
| Il reload dei moduli  | 514        |
| Le basi delle istruzioni <code>reload</code>                            | 514        |
| Un esempio di <code>reload</code>                                       | 515        |
| Riassunto   | 517        |
| Domande   | 517        |
| Risposte  | 517        |
| <b>23. I package</b> .....  | <b>519</b> |
| Importazione di package: le basi  | 519        |
| I package e il percorso di ricerca dei moduli                           | 520        |
| Il file <code>__init__.py</code> dei package                            | 520        |
| Importazione di package: un esempio                                     | 522        |
| <code>from</code> e <code>import</code> con i package                   | 523        |
| Perché usare le importazioni di package?                                | 524        |
| Una storia di tre sistemi   | 525        |
| Le importazioni relative di package                                     | 527        |
| Le modifiche di Python 3.0  | 527        |
| Le basi delle importazioni relative                                     | 528        |
| Perché le importazioni relative?  | 529        |
| Lo scopo delle importazioni relative                                    | 531        |
| Riassunto delle regole di ricerca                                       | 532        |
| Esempi di importazioni relative   | 532        |
| Riassunto   | 537        |
| Domande   | 538        |
| Risposte  | 538        |
| <b>24. Argomenti avanzati sui moduli</b> .....                          | <b>539</b> |
| Occultamento dei nomi nei moduli  | 539        |
| Minimizzare i pericoli di <code>from *: X e __all__</code>              | 539        |
| Abilitare le funzionalità future del linguaggio                         | 540        |
| Uso misto dei moduli: <code>__name__</code> e <code>__main__</code>     | 541        |
| Unit testing tramite <code>__name__</code>                              | 542        |
| Usare gli argomenti della riga di comando con <code>__name__</code>     | 543        |
| Modifiche del percorso di ricerca dei moduli                            | 545        |
| L'estensione <code>as</code> di <code>import</code> e <code>from</code> | 546        |
| I moduli sono oggetti: metaprogrammi                                    | 546        |
| Importazione di moduli tramite nomi come variabili in formato stringa   | 549        |
| Reload transitivo dei moduli  | 550        |
| Principi di progettazione dei moduli                                    | 553        |
| Trappole d'uso relative ai moduli                                       | 553        |
| Nel codice al livello più alto, l'ordine delle istruzioni conta         | 553        |

|  |     |
|--|-----|
| from copia i nomi, non le referenze                                      | 555 |
| from * può oscurare il significato delle variabili                       | 555 |
| reload potrebbe non avere effetto sulle importazioni effettuate con from | 556 |
| reload, from e test interattivo del codice                               | 556 |
| Le importazioni ricorsive tramite from potrebbero non funzionare         | 557 |
| Riassunto  | 558 |
| Domande  | 559 |
| Risposte   | 559 |
| Esercizi relativi alla parte V   | 560 |

---

## Parte VI. Classi e programmazione orientata agli oggetti

|  |            |
|--|------------|
| <b>25. Programmazione orientata agli oggetti: il quadro generale</b> ..... | <b>565</b> |
| Perché usare le classi?  | 566        |
| OOP: il quadro generale  | 567        |
| Ricerca degli Attributi ereditati  | 567        |
| Classi e istanze   | 569        |
| Chiamate ai metodi delle classi  | 570        |
| La codifica degli alberi di classi   | 571        |
| L'OOP riguarda il riutilizzo del codice                                    | 573        |
| Riassunto  | 576        |
| Domande  | 576        |
| Risposte   | 576        |
| <b>26. Codifica delle classi: le basi</b> .....                            | <b>579</b> |
| Le classi generano più oggetti istanza                                     | 579        |
| Gli oggetti classe forniscono i comportamenti di default                   | 580        |
| Gli oggetti istanza sono elementi concreti                                 | 580        |
| Un primo esempio   | 580        |
| Le classi vengono specializzate per ereditarietà                           | 582        |
| Un secondo esempio   | 583        |
| Le classi sono attributi dei moduli  | 585        |
| Le classi possono intercettare gli operatori Python                        | 586        |
| Un terzo esempio   | 587        |
| Perché usare l'overloading degli operatori?                                | 589        |
| La classe Python più semplice possibile                                    | 589        |
| Classi e dizionari   | 592        |
| Riassunto  | 594        |
| Domande  | 594        |
| Risposte   | 594        |
| <b>27. Un esempio pratico</b> .....  | <b>597</b> |
| Step 1: creazione delle istanze  | 597        |
| Codifica dei costruttori   | 598        |
| Test incrementale  | 599        |
| Step 2: definizione dei comportamenti tramite i metodi                     | 600        |

|   |            |
|---|------------|
| La codifica dei metodi  | 601        |
| Step 3: overloading degli operatori                                       | 603        |
| Visualizzazione personalizzata  | 603        |
| Step 4: specializzazione dei comportamenti tramite sottoclassi            | 604        |
| Codifica delle sottoclassi  | 605        |
| Argomenti dei metodi: la tecnica sbagliata                                | 605        |
| Argomenti dei metodi: la tecnica giusta                                   | 605        |
| Polimorfismo  | 607        |
| Ereditarietà, personalizzazione ed estensione                             | 608        |
| La grande idea delle programmazione orientata agli oggetti                | 608        |
| Step 5: personalizzazione dei costruttori                                 | 609        |
| La programmazione orientata agli oggetti è più semplice di quanto sembri  | 610        |
| Altri modi per combinare le classi  | 611        |
| Step 6: gli strumenti d'introspezione di Python                           | 613        |
| Attributi speciali delle classi   | 614        |
| Uno strumento generale di visualizzazione                                 | 615        |
| Attributi delle istanze e delle classi                                    | 616        |
| Considerazioni sui nomi degli strumenti generali                          | 617        |
| La versione finale delle nostre classi                                    | 617        |
| Step 7 (ultimo): memorizzare gli oggetti in un database                   | 618        |
| Pickles e Shelves   | 618        |
| Uso del database <code>shelve</code>                                      | 619        |
| Accesso interattivo al database   | 620        |
| Aggiornamento degli oggetti   | 622        |
| Direzioni future  | 623        |
| Riassunto   | 624        |
| Domande   | 624        |
| Risposte  | 625        |
| <b>28. I dettagli della codifica delle classi</b> .....                   | <b>627</b> |
| L'istruzione <code>class</code>   | 627        |
| Forma generale  | 627        |
| Esempio   | 628        |
| Metodi  | 630        |
| Un esempio di metodo  | 631        |
| Chiamate ai costruttori delle sovraclassi                                 | 631        |
| Altri schemi di chiamata dei metodi                                       | 632        |
| Ereditarietà  | 632        |
| Costruzione dell'albero degli attributi                                   | 633        |
| Specializzazione dei metodi ereditati                                     | 634        |
| Tecniche d'interfacciamento tra classi                                    | 634        |
| Sovraclassi astratte  | 636        |
| Le sovraclassi astratte di Python 2.6 e 3.0                               | 637        |
| Spazi di nomi: la storia completa   | 638        |
| Nomi semplici: locali se non dichiarati globali                           | 638        |
| Nomi di attributi: spazi di nomi degli oggetti                            | 639        |
| Lo "Zen" degli spazi di nomi Python: gli assegnamenti classificano i nomi | 639        |

|  |            |
|--|------------|
| I dizionari degli spazi di nomi  | 641        |
| Collegamenti tra spazi di nomi   | 644        |
| Rivisitazione delle docstring  | 646        |
| Classi e moduli  | 647        |
| Riassunto  | 648        |
| Domande  | 648        |
| Risposte   | 648        |
| <b>29. Overloading degli operatori</b>   | <b>649</b> |
| Le basi  | 649        |
| Costruttori ed espressioni: <code>__init__</code> e <code>__sub__</code>                   | 650        |
| Metodi di overloading di uso comune  | 650        |
| Indicizzazione e sezionamento: <code>__getitem__</code> e <code>__setitem__</code>         | 651        |
| Intercettare i sezionamenti  | 652        |
| Iterazione su indici: <code>__getitem__</code>   | 654        |
| Oggetti iteratori: <code>__iter__</code> e <code>__next__</code>                           | 655        |
| Iteratori definiti dall'utente   | 655        |
| Iterazioni multiple su uno stesso oggetto  | 657        |
| Appartenenza: <code>__contains__</code> , <code>__iter__</code> e <code>__getitem__</code> | 659        |
| Referenze di attributi: <code>__getattr__</code> e <code>__setattr__</code>                | 661        |
| Altri strumenti per la gestione degli attributi  | 662        |
| Emulare attributi d'istanza privati: parte 1   | 662        |
| Rappresentazione di stringa: <code>__repr__</code> e <code>__str__</code>                  | 663        |
| Addizioni destre e sul posto: <code>__radd__</code> e <code>__iadd__</code>                | 666        |
| Addizione sul posto  | 667        |
| Espressioni di chiamata: <code>__call__</code>   | 668        |
| Interfacce in stile funzione e codice basato su callback                                   | 669        |
| Confronti: <code>__lt__</code> , <code>__gt__</code> , ecc.                                | 670        |
| Il metodo <code>__cmp__</code> della 2.6 (rimosso dalla 3.0)                               | 671        |
| Test Booleani: <code>__bool__</code> e <code>__len__</code>                                | 671        |
| Distruzione di oggetti: <code>__del__</code>   | 673        |
| Riassunto  | 674        |
| Domande  | 675        |
| Risposte   | 675        |
| <b>30. Principi di progettazione delle classi</b>  | <b>677</b> |
| Python e OOP   | 677        |
| Overloading tramite segnatura delle chiamate (o no)  | 678        |
| OOP ed ereditarietà: relazioni "è un"  | 678        |
| OOP e composizione: relazioni "ha un"  | 680        |
| Rivisitazione dei flussi di dati   | 682        |
| OOP e delega: oggetti "wrapper"  | 685        |
| Attributi pseudoprivati delle classi   | 686        |
| Generalità sull'espansione dei nomi  | 687        |
| Perché usare attributi pseudo privati?   | 687        |
| I metodo sono oggetti legati o non legati  | 689        |
| In Python 3.0 i metodi sono funzioni   | 691        |

|   |            |
|---|------------|
| Metodi legati e altri oggetti chiamabili  | 692        |
| Ereditarietà multipla: classi mixin   | 695        |
| La codifica delle classi mixin  | 696        |
| Le classi sono oggetti: costruttori generici  | 704        |
| Perché i costruttori?   | 705        |
| Altri argomenti legati alla progettazione   | 706        |
| Riassunto   | 707        |
| Domande   | 707        |
| Risposte  | 707        |
| <b>31. Argomenti avanzati sulle classi</b> .....                                    | <b>709</b> |
| Estendere i tipi precostituiti  | 709        |
| Estensione tramite inglobamento   | 709        |
| Estensione tramite sottoclassamento   | 710        |
| Le classi di nuovo stile  | 713        |
| Le differenze introdotte dalle classi di nuovo stile                                | 714        |
| Modifica nel modello dei tipi   | 714        |
| Modifiche all'ereditarietà a diamante   | 718        |
| Le nuove funzionalità delle classi di nuovo stile                                   | 722        |
| Slot  | 722        |
| Proprietà delle classi  | 726        |
| __getattr__ e i descrittori   | 728        |
| Metaclassi  | 728        |
| Metodi statici e di classe  | 729        |
| Perché i metodi speciali?   | 729        |
| I metodi statici nella 2.6 e nella 3.0  | 730        |
| Alternative ai metodi statici   | 731        |
| Uso dei metodi statici e di classe  | 732        |
| Contare le istanze con un metodi statici  | 734        |
| Contare le istanze con i metodi di classe   | 735        |
| Decoratori e metaclassi (prima parte)   | 737        |
| Decoratori di funzione: le basi   | 737        |
| Un primo esempio di decoratore  | 738        |
| Decoratori di classe e metaclassi   | 739        |
| I dettagli  | 740        |
| Trappole di programmazione sulle classi   | 741        |
| La modifica degli attributi delle classi può avere effetti secondari                | 741        |
| Anche la modifica degli attributi mutabili delle classi può avere effetti secondari | 742        |
| Ereditarietà multipla: l'ordine è significativo                                     | 743        |
| Metodi, classi e scopi annidati   | 744        |
| Classi basate su delega nella 3.0: __getattr__ e i precostituiti                    | 746        |
| Annidare troppo   | 746        |
| Riassunto   | 747        |
| Domande   | 747        |
| Risposte  | 747        |
| Esercizi relativi alla Parte VI   | 748        |



---

## Parte VII. Eccezioni e strumenti di sviluppo

|  |            |
|--|------------|
| <b>32. Le basi delle eccezioni</b> .....                                 | <b>757</b> |
| Perché utilizzare le eccezioni?  | 757        |
| I ruoli delle eccezioni  | 758        |
| Le eccezioni in breve  | 759        |
| Il gestore di default  | 759        |
| Come intrappolare le eccezioni   | 760        |
| Generazione esplicita delle eccezioni                                    | 761        |
| Eccezioni definite dall'utente   | 761        |
| Azioni terminali   | 762        |
| Riassunto  | 764        |
| Domande  | 765        |
| Risposte   | 765        |
| <br>   |            |
| <b>33. I dettagli di codifica delle eccezioni</b> .....                  | <b>767</b> |
| L'istruzione <code>try/except/else</code>                                | 767        |
| Le proposizioni dell'istruzione <code>try</code>                         | 768        |
| La proposizione <code>try/else</code>                                    | 771        |
| Esempio: comportamento di default  | 771        |
| Esempio: intrappolare le eccezioni precostituite                         | 772        |
| L'istruzione <code>try/finally</code>                                    | 773        |
| Esempio: azioni terminali con <code>try/finally</code>                   | 774        |
| La forma unificata <code>try/except/finally</code>                       | 775        |
| Sintassi della forma unificata   | 776        |
| Combinare <code>finally</code> e <code>except</code> tramite annidamento | 777        |
| Un esempio di <code>try</code> unificata                                 | 777        |
| L'istruzione <code>raise</code>  | 779        |
| Propagazione delle eccezioni con <code>raise</code>                      | 780        |
| La concatenazione delle eccezioni in Python 3.0: <code>raise from</code> | 780        |
| L'istruzione <code>assert</code>   | 781        |
| Esempio: intrappolare vincoli (ma non errori!)                           | 782        |
| I gestori di contesto <code>with/as</code>                               | 782        |
| Uso di base  | 783        |
| Il protocollo di gestione dei contesti                                   | 784        |
| Riassunto  | 786        |
| Domande  | 786        |
| Risposte   | 787        |
| <br>   |            |
| <b>34. Gli oggetti eccezione</b> .....                                   | <b>789</b> |
| Eccezioni: ritorno al futuro   | 790        |
| Le eccezioni basate sulle stringhe non esistono più!                     | 790        |
| Eccezioni basate sulle classi  | 790        |
| Come codificare le eccezioni basate su classi                            | 791        |
| Perché le gerarchie di eccezioni?  | 793        |
| Le eccezioni precostituite   | 795        |

|   |            |
|---|------------|
| Le categorie di eccezioni precostituite                       | 796        |
| Visualizzazione e dati di stato di default                    | 797        |
| Messaggi personalizzati                                       | 798        |
| Dati e comportamenti personalizzati                           | 799        |
| Fornire i dettagli delle eccezioni                            | 799        |
| Definizione di metodi   | 800        |
| Riassunto   | 801        |
| Domande   | 801        |
| Risposte  | 801        |
| <b>35. Progettazione e uso delle eccezioni .....</b>          | <b>803</b> |
| Gestori annidati  | 803        |
| Esempio: flussi di controllo annidati                         | 804        |
| Esempio: annidamento sintattico                               | 805        |
| Idiomi di codifica  | 806        |
| Le eccezioni non sono sempre errori                           | 807        |
| Le funzioni possono segnalare condizioni tramite <b>raise</b> | 807        |
| Chiusura dei file e delle connessioni lato server             | 808        |
| Debug tramite <b>try</b> esterne                              | 809        |
| Eseguire test dall'interno del processo                       | 809        |
| I dettagli di <b>sys.exc_info</b>                             | 810        |
| Consigli e trappole di progettazione                          | 811        |
| Cosa racchiudere  | 811        |
| Intrappolare troppo: evitare <b>try</b> vuote                 | 812        |
| Intrappolare troppo poco: usate categorie basate su classi    | 813        |
| Riassunto del linguaggio base                                 | 814        |
| La cassetta degli attrezzi Python                             | 814        |
| Strumenti di sviluppo per progetti complessi                  | 815        |
| Riassunto   | 818        |
| Domande   | 819        |
| Risposte  | 819        |
| Esercizi relativi alla Parte VII                              | 819        |

---

## Parte VIII. Argomenti avanzati

|   |            |
|---|------------|
| <b>36. Stringhe Unicode e di byte .....</b>         | <b>823</b> |
| Le modifiche alle stringhe introdotte da Python 3.0 | 823        |
| Concetti base sulle stringhe                        | 824        |
| Schemi di codifica dei caratteri                    | 824        |
| I tipi stringa previsti da Python                   | 826        |
| File di testo e binari                              | 827        |
| Esempi d'uso delle stringhe di Python 3.0           | 829        |
| Letterali e proprietà di base                       | 829        |
| Conversioni   | 830        |
| Codifica delle stringhe Unicode                     | 831        |
| Codifica del testo ASCII                            | 831        |

|   |            |
|---|------------|
| Codifica di testo non ASCII   | 832        |
| Codifica e decodifica di testo non ASCII                                      | 832        |
| Altre tecniche di codifica Unicode  | 833        |
| Conversione tra codifiche diverse   | 835        |
| Codifica delle stringhe Unicode in Python 2.6                                 | 835        |
| Dichiarazioni del set di caratteri Unicode nei file di codice sorgente Python | 838        |
| Uso degli oggetti <b>bytes</b> di Python 3.0                                  | 839        |
| Metodi  | 839        |
| Operazioni di sequenza  | 840        |
| Altri modi per creare oggetti <b>bytes</b>                                    | 841        |
| Mischiare stringhe e byte   | 841        |
| Uso degli oggetti <b>bytearray</b> di Python 3.0 (e della 2.6)                | 842        |
| Uso dei file di testo e binari  | 845        |
| Processamento di base dei file di testo                                       | 845        |
| Modalità testo o binaria in Python 3.0  | 846        |
| Tipi e contenuti disomogenei  | 847        |
| Uso dei file Unicode  | 848        |
| Lettura e scrittura di dati Unicode in Python 3.0                             | 848        |
| La gestione del BOM in Python 3.0   | 850        |
| I file Unicode in Python 2.6  | 852        |
| Altre modifiche agli strumenti per le stringhe di Python 3.0                  | 853        |
| Il modulo <b>re</b>   | 853        |
| Il modulo <b>struct</b>   | 854        |
| Il modulo <b>pickle</b>   | 855        |
| Strumenti XML   | 856        |
| Riassunto   | 859        |
| Domande   | 859        |
| Risposte  | 859        |
| <br>  |            |
| <b>37. Gestione degli attributi</b> .....                                     | <b>863</b> |
| Perché gestire gli attributi?   | 863        |
| Come eseguire codice durante l'accesso agli attributi                         | 864        |
| Proprietà   | 864        |
| Le basi   | 864        |
| Un primo esempio  | 865        |
| Attributi dinamici  | 866        |
| Codifica delle proprietà tramite decoratori                                   | 867        |
| I descrittori   | 868        |
| Le basi   | 869        |
| Un primo esempio  | 871        |
| Attributi dinamici  | 873        |
| Uso dei dati di stato nei descrittori   | 873        |
| Il legame tra proprietà e descrittori   | 875        |
| <u>__getattr__</u> e <u>__getattribute__</u>                                  | 876        |
| Le basi   | 877        |
| Un primo esempio  | 878        |
| Attributi dinamici  | 880        |

|  |            |
|--|------------|
| Confronto tra <code>__getattr__</code> e <code>__getattribute__</code> | 881        |
| Confronto tra le tecniche di gestione degli attributi                  | 882        |
| Intercettare gli accessi agli attributi delle operazioni precostituite | 884        |
| Ancora sui manager basati sulla delega                                 | 888        |
| Esempio: validazione degli attributi                                   | 891        |
| Validazione tramite le proprietà                                       | 891        |
| Validazione tramite i descrittori                                      | 893        |
| Validazione tramite <code>__getattr__</code>                           | 894        |
| Validazione tramite <code>__getattribute__</code>                      | 895        |
| Riassunto  | 896        |
| Domande  | 896        |
| Risposte   | 897        |
| <b>38. I decorator</b> .....   | <b>899</b> |
| Cos'è un decoratore?   | 899        |
| Gestione delle chiamate e delle istanze                                | 900        |
| Gestione delle funzioni e delle classi                                 | 900        |
| Uso e definizione dei decoratori                                       | 900        |
| Perché i decoratori?   | 901        |
| Le basi  | 902        |
| Decoratori di funzione   | 902        |
| Decoratori di classe   | 905        |
| Annidamento dei decoratori   | 908        |
| Gli argomenti dei decoratori   | 909        |
| I decoratori possono gestire anche oggetti funzione e classe           | 910        |
| Codifica dei decoratori di funzione                                    | 910        |
| Tracciatura delle chiamate   | 911        |
| Opzioni di memorizzazione dei dati di stato                            | 912        |
| Errori con le classi: decorazione dei metodi                           | 915        |
| Temporizzazione delle chiamate   | 920        |
| Passaggio degli argomenti ai decoratori                                | 921        |
| Codifica dei decoratori di classe                                      | 924        |
| Classi singleton   | 924        |
| Tracciare le interfacce delle istanze                                  | 925        |
| Errori con le classi II: memorizzare più istanze                       | 929        |
| Decoratori e funzioni manager  | 930        |
| Perché i decoratori (ancora)?  | 931        |
| Gestione diretta delle funzioni e delle classi                         | 932        |
| Esempio: attributi privati e pubblici                                  | 935        |
| Implementazione di attributi privati                                   | 935        |
| Dettagli d'implementazione 1   | 937        |
| Generalizzazione alle dichiarazioni pubbliche                          | 938        |
| Dettagli d'implementazione 2   | 940        |
| Problemi aperti  | 941        |
| Python non è controllo   | 944        |
| Esempio: validazione degli argomenti delle funzioni                    | 945        |
| L'obiettivo  | 945        |

|   |            |
|---|------------|
| Versione base: appartenenza degli argomenti posizionali a un intervallo di valori | 946        |
| Generalizzazione agli argomenti keyword e ai default                              | 948        |
| Dettagli d'implementazione  | 950        |
| Problemi aperti   | 952        |
| Argomenti dei decoratori e annotazioni di funzioni                                | 952        |
| Altre applicazioni: test dei tipi (se insistete!)                                 | 954        |
| Riassunto   | 955        |
| Domande   | 955        |
| Risposte  | 955        |
| <b>39. Le metaclassi</b>  | <b>959</b> |
| Usare o non usare le metaclassi?  | 959        |
| Livelli crescenti di magia  | 960        |
| Lo svantaggio delle funzioni helper   | 961        |
| Metaclassi e decoratori: parte 1  | 963        |
| Il modello delle metaclassi   | 964        |
| Le classi sono istanze di <code>type</code>                                       | 964        |
| Le metaclassi sono sottoclassi di <code>type</code>                               | 967        |
| Il protocollo dell'istruzione <code>class</code>                                  | 967        |
| La dichiarazione delle metaclassi   | 968        |
| La codifica delle metaclassi  | 969        |
| Una metaclassa di base  | 969        |
| Personalizzazione della costruzione e della inizializzazione                      | 970        |
| Altre tecniche di codifica delle metaclassi                                       | 971        |
| Istanze ed ereditarietà   | 974        |
| Esempio: aggiungere metodi alle classi  | 975        |
| Estensione manuale  | 975        |
| Estensione tramite metaclassi   | 976        |
| Metaclassi e decoratori: parte 2  | 977        |
| Esempio: applicazione di un decoratore a tutti i metodi di una classe             | 980        |
| Tracciatura manuale tramite decoratori  | 980        |
| Tracciatura tramite metaclassi e decoratori                                       | 981        |
| Applicazione ai metodi di un qualsiasi decoratore                                 | 982        |
| Metaclassi e decoratori: parte 3  | 984        |
| Riassunto   | 985        |
| Domande   | 986        |
| Risposte  | 986        |

---

## Parte IX. Appendici

|  |             |
|--|-------------|
| <b>A. Installazione e configurazione</b>                   | <b>991</b>  |
| <b>B. Soluzioni degli esercizi riassuntivi delle parti</b> | <b>1001</b> |
| <b>Indice analitico</b>                                    | <b>1037</b> |