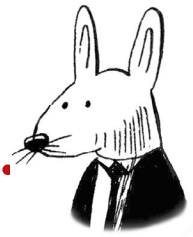


Для кого написано цю книжку

Книжка передбачає, що читач має щонайменше один рік комерційного досвіду розробки програмного забезпечення з використанням популярної об'єктно-орієнтованої мови програмування, такої як Java. У прикладах використовується Scala як навчальна мова, але це не книжка для вивчення Scala. Попередні знання Scala або функційного програмування не потрібні.

Структура книжки

Книжка складається з трьох частин. У першій частині закладено підвалини. Ми дізнаємося про інструменти та методи, які повсюдно використовуються у функційному програмуванні (ФП). У розділі 1 ми обговоримо, як навчатися ФП за допомогою цієї книжки. У розділі 2 ви побачите різницю між чистими й нечистими функціями. У розділі 3 ви ознайомитеся з незмінюваними значеннями. Нарешті, у розділі 4 вам стане очевидним, що чисті функції — це просто незмінювані значення, і ви зможете дослідити всі суперможливості, які відкриваються завдяки цьому факту.

У другій частині книжки ми будемо використовувати лише незмінювані значення та чисті функції для розв'язування прикладних задач. У розділі 5 ми розповімо про з найважливішу функцію ФП і покажемо, як вона допомагає створювати послідовні значення (і програми) у стислий і зручний для читання спосіб. У розділі 6 ви дізнаєтеся, як створювати послідовні програми, які можуть повертати помилки. У розділі 7 ми поговоримо про функційний дизайн програмного забезпечення. Розділ 8 навчить вас, як приборкати зовнішній світ нечистого коду з побічними ефектами та створювати безпечні й функційні програми. Потім ви ознайомитеся з потоками

й потоковими системами в розділі 9. Ми будемо створювати потоки з сотень тисяч елементів завдяки функційному підходу. У розділі 10 ми нарешті створимо кілька функційних і безпечних паралельних програм.

У третій частині ми реалізуємо функційними методами реальний застосунок, який використовуватиме Wikidata як джерело даних. На цьому прикладі ми продемонструємо все, чого ми навчилися в попередніх частинах. У розділі 11 ми створимо модель незмінюваних даних і використаємо відповідні типи, у тому числі ввід-вивід, для інтеграції з Wikidata, застосуємо кешування й багатопотоковість, щоб пришвидшити роботу застосунка. Ми обернемо всі ці завдання в чисті функції, а також покажемо, як ваші знання з об'єктно-орієнтованого програмування можуть стати вам у пригоді у функційному світі. У розділі 12 ви побачите, як тестувати застосунок, який ми розробили у розділі 11, і наскільки легко його підтримувати навіть за значних змін у вимогах.

Книжка завершується фінальним набором вправ для закріплення отриманих знань із функційного програмування.

Про код

Ця книжка містить багато прикладів вихідного коду як у вигляді нумерованих лістингів, так і у вигляді звичайного тексту. В обох випадках вихідний код оформлений моноширинним шрифтом для наочного відокремлення від звичайного тексту. Подекуди ми виділяємо код напівжирним шрифтом, щоб позначити елементи, які змінилися порівняно з попередніми кроками у розділі, наприклад, коли до рядка коду додається нова функція.

У багатьох випадках вихідний код було переформатовано: ми додали розриви рядків і переробили відступи, щоб умістити його на сторінці. Лістинги супроводжують анотації, які пояснюють важливі концепції.

Ви можете отримати виконувані фрагменти коду з онлайн-версії цієї книжки на сайті https://livebook.manning.com/book/grokking-functional-programming. Повний код для прикладів, наведених у книжці, можна завантажити з веб-сайту видавництва Manning за посиланням https://www.manning.com/books/grokking-functional-programming та з GitHub за посиланням https:// github.com/miciek/grokkingfp-examples. Усі ресурси книжки, включно з бонусними матеріалами, доступні на сайті https://michalplachta.com/book/.

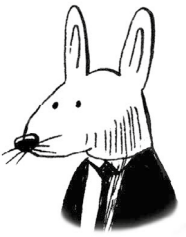
Дискусійний форум liveBook

Купивши книжку «Функційне програмування», ви отримуєте безкоштовний доступ до liveBook, платформи для онлайн-читання видавництва

Manning. Ексклюзивні функції форуму liveBook дають змогу коментарі до книжки в цілому або до окремих розділів чи параграфів. Ви можете робити нотатки для себе, ставити технічні запитання і відповідати на них, а також отримувати допомогу від автора та інших користувачів. Щоб отримати доступ до форуму, перейдіть за покликанням https://livebook.manning.com/book/grokking-functional-programming/discussion. Ви також можете дізнатися більше про форуми Manning і правила поведінки на них за покликанням https://livebook.manning.com/discussion.

Як жест прихильності до читача, видавництво Manning надає простір для діалогу між окремими читачами, а також між читачами й автором. Автор не має жодних зобов'язань щодо певного обсягу участі, його внесок у форум залишається добровільним (і неоплачуваним). Ми радимо вам спробувати поставити автору кілька складних запитань, щоб викликати його зацікавлення. Форум та архіви попередніх обговорень будуть доступні на сайті видавництва, доки книжка виходить друком.

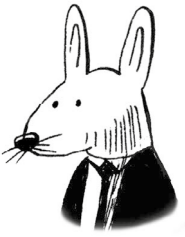
про автора



Міхал Плахта — досвідчений інженер-програміст і активний учасник спільноти функційного програмування. Він регулярно виступає на конференціях, проводить воркшопи, організовує зустрічі та веде блоги про створення підтримуваного програмного забезпечення.

Частина 1

Інструментарій ФП



Перша частина книжки «Ґрокаємо функційне програмування» закладає основи. Ви дізнаєтеся про інструменти та методи, які повсюдно використовуються у функційному програмуванні. Все, про що ви дізнаєтеся в цій частині, буде використовуватися в наступних розділах і впродовж усієї вашої кар'єри.

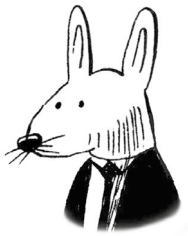
У **розділі 1** ми обговоримо основи і з'ясуємо, яких підходів до викладу дотримано в цій книжці. Ми налаштуємо середовище, напишемо код і розв'яжемо кілька перших задач.

У **розділі 2** ми обговоримо різницю між чистими та нечистими функціями. Ми використаємо кілька важливих прикладів, щоб показати небезпеки та фрагменти функційного коду, які допомагають їх мінімізувати.

У **третьому розділі** ми познайомимо вас із партнером по танцю чистих функцій: незмінюваним значенням. Ми розповімо, чому одне з них не може жити без іншого, і чому вони вдвох визначають, що таке функційне програмування.

Нарешті, у **розділі 4** ми покажемо, чому чисті функції є просто значеннями, і продемонструємо всі суперможливості, які ми отримуємо від цього факту. Це дозволить нам з'єднати всі крапки над «і» й зібрати наш перший повноцінний функційний інструментарій.

Вивчаємо функційне програмування 1



У цьому розділі

Ви дізнаєтеся:

- для кого призначено цю книжку,
- що таке функція,
- чим корисне функційне програмування,
- як встановити необхідні інструменти,
- як користуватися цією книжкою.

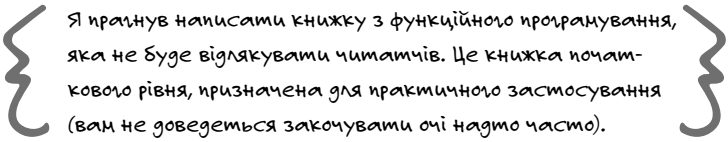
«Я можу лише навести конкретні приклади
й дати вам змогу робити висновки самостійно»

— Річард Геммінґ, «Вчимося вчитися»

Можливо, ви взялися за цю книжку тому, що...

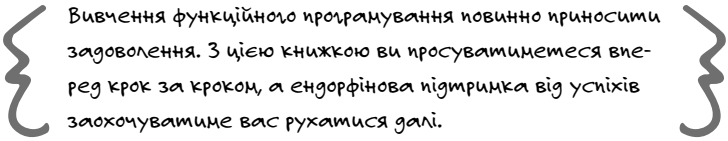
Вас цікавить функційне програмування. 1

Ви чули про функційне програмування, прочитали статтю у Вікіпедії та переглянули кілька книжок. Можливо, від математичних пояснень до коду ви закочували очі, але все одно не втратили цікавості.



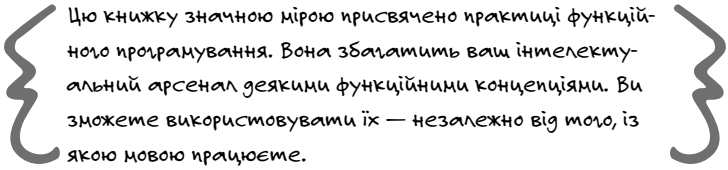
Ви вже намагалися вивчити функційне програмування раніше. 2

Ви не раз бралися за вивчення функційного програмування, але все одно достеменно не розумієте його. Тільки-но ви освоїли одну концепцію — як за рогом чигає наступна. І перш ніж бодай наблизитися до неї, вам слід дізнатися ще безліч різних речей.



Ви все ще вагаєтеся. 3

Ви вже багато років програмуєте в об'єктно-орієнтованій або імперативній мові програмування. Ви відчули кайф від функційного програмування, прочитали кілька постів у блогах і трохи спробували кодити. Але ви все ще не розумієте, як воно може полегшити програмування.



Можливо, я щось не назвав?

Якими б не були ваші причини, у цій книжці ми спробуємо зовсім інший підхід: **навчання через експеримент і гру**. Ви будете ставити питання і знаходити відповіді на них за допомогою коду. Це допоможе вам зростати професійно. Сподіваюсь, вам сподобається.

Що треба знати, перш ніж почати?

Ми припускаємо, що ви розробляли програмне забезпечення будь-якою з популярних мов, таких як Java, C++, C#, JavaScript або Python. Це дуже туманне визначення, тому ось кілька коротких чеклістів, які допоможуть нам уникнути непорозумінь.

Вам буде комфортно навчатися, якщо

- Ви знайомі з базовими поняттями об'єктно-орієнтованого програмування, такими як класи та об'єкти.
- Ви вмієте читати та розуміти подібний код:

```
class Book {
  private String title;                          1
  private List<Author> authors;                  1

  public Book(String title) {                    2
    this.title = title;
    this.authors = new ArrayList<Author>();
  }

  public void addAuthor(Author author) {         3
    this.authors.add(author);
  }
}
```

1 Книжка має назву та список об'єктів Author.
2 Конструктор: Створює новий об'єкт Book із назвою та без списку Author
3 Додає автора для цього екземпляра Book

Ви отримаєте максимальну користь, якщо

- У вас виникали проблеми зі стабільністю, тестованістю, регресією або інтеграцією ваших програмних модулів.
- У вас виникали проблеми з налагодженням подібного коду:

```
public void makeSoup(List<String> ingredients) {   1
  if(ingredients.contains("water")) {
    add("water");
  } else throw new NotEnoughIngredientsException();
  heatUpUntilBoiling();
  addVegetablesUsing(ingredients);
  waitMinutes(20);
}
```

1 Напевно, це не найкращий суп, що ви будь-коли куштували...

Вам не потрібно:

- Бути експертом у об'єктно-орієнтованому програмуванні
- Бути експертом із Java/C++/C#/Python
- Щось знати про будь-яку функційну мову програмування, як-от Kotlin, Scala, F#, Rust, Clojure або Haskell.

Як виглядають функції

Перейдімо до коду без зволікань! Ми ще не встановили всі необхідні інструменти, але це нас не зупинить, еге ж?

Перед вами набір різних функцій. Усі вони мають дещо спільне: вони отримують деякі значення на вході, щось роблять і, можливо, повертають значення на виході. Отже, дивіться:

```
public static int add(int a, int b) {                    1
 return a + b;
}

public static char getFirstCharacter(String s) {         2
 return s.charAt(0);
}

public static int divide(int a, int b) {                  3
 return a / b;
}

public static void eatSoup(Soup soup) {
 // TODO: алгоритм «поїдання супу»                        4
}
```

1. Отримує два цілі числа int, додає їх і повертає суму
2. Отримує рядок і повертає його перший символ
3. Отримує два входи, ділить перший на другий і повертає результат
4. Отримує об'єкт Soup, щось робить із ним і не повертає нічого

Навіщо нам всі ці public statics?

Вас, напевно, дивує модифікатор public static у кожному визначенні. Це зроблено навмисно. Всі функції, які ми використовуємо в цій книжці, є статичними (тобто, для їх виконання не потрібен екземпляр об'єкта). Вони вільні — їх може викликати будь-який користувач і з будь-якого місця, якщо він вводить потрібні вхідні параметри. Вони працюють лише з тими даними, які надає користувач, і більше ні з чим.

Це, звичайно, має деякі серйозні наслідки, які ми розглянемо далі в цій книжці. Поки слід пам'ятати, що коли

ми говоримо про **функцію**, ми маємо на увазі загальнодоступну статичну функцію, яку можна викликати з будь-якого місця.

Коротка вправа

Реалізуйте дві наведені нижче функції:

```
public static int increment(int x) {
  // TODO
}
public static String concatenate(String a, String b) {
  // TODO
}
```

Відповіді:

```
return x + 1;
return a + b;
```