

Spis treści

Przedmowa	9
Podziękowania	11
O książce	13
O autorze	17
CZĘŚĆ I. MYŚL FUNKCYJNE.....	19
Rozdział 1. Przechodzenie na model funkcyjny	21
1.1. Czy programowanie funkcyjne może być pomocne?	23
1.2. Czym jest programowanie funkcyjne?	24
1.2.1. Programowanie funkcyjne jest deklaratywne	26
1.2.2. Czyste funkcje i problemy z efektami ubocznymi	27
1.2.3. Przezroczystość referencyjna i możliwość podstawiania	31
1.2.4. Zachowywanie niemodyfikowalności danych	33
1.3. Zalety programowania funkcyjnego	34
1.3.1. Ułatwianie podziału złożonych zadań	34
1.3.2. Przetwarzanie danych za pomocą płynnych łańcuchów wywołań	36
1.3.3. Radzenie sobie ze złożonością aplikacji asynchronicznych	38
1.4. Podsumowanie	40
Rozdział 2. Operacje wyższego poziomu w JavaScriptcie	41
2.1. Dlaczego JavaScript?	42
2.2. Programowanie funkcyjne a programowanie obiektowe	42
2.2.1. Zarządzanie stanem obiektów w JavaScriptcie	49
2.2.2. Traktowanie obiektów jak wartości	49
2.2.3. Głębokie zamrażanie potencjalnie zmiennych elementów	52
2.2.4. Poruszanie się po grafach obiektów i ich modyfikowanie za pomocą soczewek	54
2.3. Funkcje	56
2.3.1. Funkcje jako pełnoprawne obiekty	56
2.3.2. Funkcje wyższego poziomu	57
2.3.3. Sposoby uruchamiania funkcji	59
2.3.4. Metody używane dla funkcji	61
2.4. Domknięcia i zasięg	62
2.4.1. Problemy z zasięgiem globalnym	64
2.4.2. Zasięg funkcji w JavaScriptcie	65
2.4.3. Zasięg pseudobloku	66
2.4.4. Praktyczne zastosowania domknięć	67
2.5. Podsumowanie	70

CZĘŚĆ II. WKROCZ W ŚWIAT PROGRAMOWANIA FUNKCYJNEGO71

Rozdział 3. Niewielka liczba struktur danych i wiele operacji	73
3.1. Przepływ sterowania w aplikacji	74
3.2. Łączenie metod w łańcuch	75
3.3. Łączenie funkcji w łańcuch	76
3.3.1. Wyrażenia lambda	77
3.3.2. Przekształcanie danych za pomocą operacji <code>_map</code>	78
3.3.3. Pobieranie wyników za pomocą operacji <code>_reduce</code>	80
3.3.4. Usuwanie niepotrzebnych elementów za pomocą funkcji <code>_filter</code>	84
3.4. Analizowanie kodu	85
3.4.1. Deklaratywne łańcuchy funkcji w podejściu leniwym	86
3.4.2. Dane w formacie podobnym do SQL-owego — traktowanie funkcji jak danych	90
3.5. Naucz się myśleć rekurencyjnie	91
3.5.1. Czym jest rekurencja?	92
3.5.2. Jak nauczyć się myśleć rekurencyjnie?	92
3.5.3. Rekurencyjnie definiowane struktury danych	95
3.6. Podsumowanie	98
Rozdział 4. W kierunku modularnego kodu do wielokrotnego użytku	99
4.1. Łańcuchy metod a potoki funkcji	100
4.1.1. Łączenie metod w łańcuchy	101
4.1.2. Porządkowanie funkcji w potoku	102
4.2. Wymogi dotyczące zgodności funkcji	103
4.2.1. Funkcje zgodne ze względu na typ	103
4.2.2. Funkcje i arność — argument na rzecz stosowania krotek	104
4.3. Przetwarzanie funkcji z rozwijaniem	107
4.3.1. Emulowanie fabryk funkcji	110
4.3.2. Tworzenie przeznaczonych do wielokrotnego użytku szablonów funkcji	111
4.4. Częściowe wywoływanie funkcji i wiązanie parametrów	113
4.4.1. Rozszerzanie podstawowego języka	115
4.4.2. Wiązanie funkcji wykonywanych z opóźnieniem	115
4.5. Tworzenie potoków funkcji za pomocą kompozycji	116
4.5.1. Kompozycja na przykładzie kontrolek HTML-owych	117
4.5.2. Kompozycja funkcyjna — oddzielenie opisu od przetwarzania	118
4.5.3. Kompozycja z użyciem bibliotek funkcyjnych	121
4.5.4. Radzenie sobie z kodem czystym i nieczystym	123
4.5.5. Wprowadzenie do programowania bezargumentowego	124
4.6. Zarządzanie przepływem sterowania z użyciem kombinatorów funkcji	126
4.6.1. Kombinator <code>identity</code>	126
4.6.2. Kombinator <code>tap</code>	126
4.6.3. Kombinator <code>alt</code>	127
4.6.4. Kombinator <code>seq</code>	128
4.6.5. Kombinator <code>fork</code>	128
4.7. Podsumowanie	130

Rozdział 5. Wzorce projektowe pomagające radzić sobie ze złożonością	131
5.1. Wady imperatywnej obsługi błędów	132
5.1.1. Obsługa błędów za pomocą bloków try-catch	132
5.1.2. Dlaczego w programach funkcyjnych nie należy zgłaszać wyjątków?	133
5.1.3. Problemy ze sprawdzaniem wartości null	134
5.2. Budowanie lepszego rozwiązania — funktory	135
5.2.1. Opakowywanie niebezpiecznych wartości	136
5.2.2. Funktory	138
5.3. Funkcyjna obsługa błędów z użyciem monad	140
5.3.1. Monad — od przepływu sterowania do przepływu danych	141
5.3.2. Obsługa błędów za pomocą monad Maybe i Either	145
5.3.3. Interakcje z zewnętrznymi zasobami przy użyciu monady IO	154
5.4. Monadyczne łańcuchy i kompozycje	157
5.5. Podsumowanie	163

CZĘŚĆ III. ROZWIJANIE UMIEJĘTNOŚCI W ZAKRESIE PROGRAMOWANIA FUNKCYJNEGO 165

Rozdział 6. Zabezpieczanie kodu przed błędami	167
6.1. Wpływ programowania funkcyjnego na testy jednostkowe	168
6.2. Problemy z testowaniem programów imperatywnych	169
6.2.1. Trudność identyfikowania i wyodrębniania zadań	170
6.2.2. Zależność od współużytkowanych zasobów prowadzi do niespójnych wyników	171
6.2.3. Zdefiniowana kolejność wykonywania operacji	172
6.3. Testowanie kodu funkcyjnego	173
6.3.1. Traktowanie funkcji jak czarnych skrzynek	173
6.3.2. Koncentracja na logice biznesowej zamiast na przepływie sterowania	174
6.3.3. Oddzielanie czystego kodu od nieczystego za pomocą monadycznej izolacji	176
6.3.4. Tworzenie atrap zewnętrznych zależności	178
6.4. Przedstawianie specyfikacji w testach opartych na cechach	180
6.5. Pomiar efektywności testów na podstawie pokrycia kodu	186
6.5.1. Pomiar efektywności testów kodu funkcyjnego	187
6.5.2. Pomiar złożoności kodu funkcyjnego	190
6.6. Podsumowanie	193

Rozdział 7. Optymalizacje funkcyjne	195
7.1. Praca funkcji na zapleczu	196
7.1.1. Rozwijanie funkcji a kontekst funkcji na stosie	198
7.1.2. Wyzwania związane z kodem rekurencyjnym	200
7.2. Odraczanie wykonywania funkcji za pomocą leniwego wartościowania	202
7.2.1. Unikanie obliczeń dzięki kombinatorowi funkcyjnemu alt	203
7.2.2. Wykorzystanie syntezy wywołań	204
7.3. Wywoływanie kodu wtedy, gdy jest potrzebny	206
7.3.1. Memoizacja	207
7.3.2. Memoizacja funkcji o dużych wymaganiach obliczeniowych	207

7.3.3.	<i>Wykorzystanie rozwijania funkcji i memoizacji</i>	210
7.3.4.	<i>Dekompozycja w celu zastosowania memoizacji do maksymalnej liczby komponentów</i>	211
7.3.5.	<i>Stosowanie memoizacji do wywołań rekurencyjnych</i>	212
7.4.	<i>Rekurencja i optymalizacja wywołań ogonowych</i>	213
7.4.1.	<i>Przekształcanie wywołań nieogonowych w ogonowe</i>	215
7.5.	<i>Podsumowanie</i>	218
Rozdział 8.	<i>Zarządzanie asynchronicznymi zdarzeniami i danymi</i>	219
8.1.	<i>Problemy związane z kodem asynchronicznym</i>	220
8.1.1.	<i>Tworzenie związanych z czasem zależności między funkcjami</i>	221
8.1.2.	<i>Powstawanie piramidy wywołań zwrotnych</i>	222
8.1.3.	<i>Styl oparty na przekazywaniu kontynuacji</i>	224
8.2.	<i>Pełnoprawne operacje asynchroniczne oparte na obietnicach</i>	227
8.2.1.	<i>Łańcuchy metod wykonywanych w przyszłości</i>	230
8.2.2.	<i>Kompozycja operacji synchronicznych i asynchronicznych</i>	235
8.3.	<i>Leniwe generowanie danych</i>	237
8.3.1.	<i>Generatory i rekurencja</i>	239
8.3.2.	<i>Protokół iteratorów</i>	241
8.4.	<i>Programowanie funkcyjne i reaktywne z użyciem biblioteki RxJS</i>	242
8.4.1.	<i>Dane jako obserwowalne sekwencje</i>	242
8.4.2.	<i>Programowanie funkcyjne i reaktywne</i>	243
8.4.3.	<i>RxJS i obietnice</i>	246
8.5.	<i>Podsumowanie</i>	246
Dodatek.	<i>Biblioteki JavaScriptu używane w książce</i>	249
Skorowidz		253