

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

ActionScript 3.0. Biblia

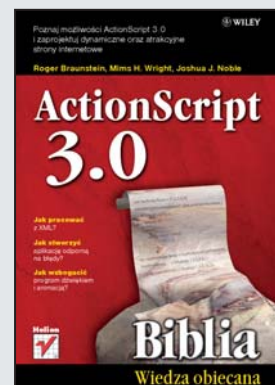
Autor: Roger Braunstein, Mims H. Wright,
Joshua J. Noble

Tłumaczenie: Sebastian Rataj

ISBN: 978-83-246-1939-9

Tytuł oryginału: [ActionScript 3.0 Bible](#)

Format: 172x245, stron: 752



Poznaj możliwości ActionScript 3.0 i zaprojektuj dynamiczne oraz atrakcyjne strony internetowe

- Jak pracować z XML?
- Jak stworzyć aplikację odporną na błędy?
- Jak wzbogacić program dźwiękiem i animacją?

Język programowania ActionScript wykorzystywany jest do tworzenia treści dla programu Flash Player. Jest on niezbędny wszędzie tam, gdzie trzeba tworzyć dynamiczne aplikacje Flash, reagujące na działanie użytkownika, nadające się do ponownego użycia i łatwe w dostosowaniu do nowych okoliczności. Wersja ActionScript 3.0 została stworzona od podstaw z wykorzystaniem najnowszej specyfikacji ECMA. W aktualnej wersji tego języka wprowadzono wiele nowych funkcji, m.in. nowy sposób zarządzania typami wyświetlania, obsługę błędów czasu wykonania, zakończenia metod i wewnętrzny model zdarzeń.

Książka „ActionScript 3.0. Biblia” obszernie i wyczerpująco opisuje ten obiektowy język programowania, służący do budowania nowoczesnych aplikacji internetowych. Znajdziesz tu zarówno niezbędną wiedzę teoretyczną, jak i praktyczne przykłady, ilustrujące chociażby programowanie z wykorzystaniem klas, używanie tablic i obiektów czy obsługę błędów. Z tym podręcznikiem nauczysz się tworzyć interaktywną zawartość Flasha oraz aplikacje Flex, a także pracować z dźwiękiem, animacją i danymi zewnętrznymi. Zdobędziesz wiedzę, która pozwoli Ci na zrealizowanie wszelkich pomysłów dotyczących Twojego programu.

- Rozpoczęcie pracy z ActionScript 3.0
- Stosowanie zmiennych
- Praca z operatorami
- Programowanie z wykorzystaniem klas
- Metody i funkcje
- Konwersje liczbowe
- Praca z XML
- Wyrażenia regularne
- Pola tekstowe i formatowanie tekstu
- Korzystanie z timerów
- Obsługa błędów
- Praca z danymi zewnętrznymi
- Wzbogacanie programu dźwiękiem i wideo
- Wdrażanie programu

Poznaj ActionScript i wykorzystaj pełnię jego możliwości!

Spis treści

O autorach	23
Podziękowania	25
Wstęp	27
Część I Rozpoczęcie pracy z ActionScript 3.0	45
Rozdział 1. Wprowadzenie do ActionScript 3.0	47
Czym jest ActionScript?	47
Do czego używać ActionScript 3.0?	48
Co nowego w ActionScript 3.0?	48
Lista wyświetlania	49
Błędy czasu wykonywania	49
Kontrola typów danych w czasie wykonywania	49
Domknięcia metod	50
Wewnętrzny model zdarzeń	50
Wyrażenia regularne	50
E4X	50
Podsumowanie	51
Rozdział 2. Podstawy języka ActionScript 3.0	53
Stosowanie zmiennych	53
Anatomia deklaracji zmiennej	54
Stałe w tym zmieniającym się świecie	54
Stosowanie literalów	55
Stosowanie kontroli dostępu	55
Zasięg	56
Typy zasięgu	57
Łącuch zasięgu	59
Praca z zasięgiem	61
Typy danych	62
Deklarowanie typów	62
Używanie zmiennych bez typu z typem danych wieloznacznika (*)	63
Praca z operatorami	63
Operatory jedno- i dwuargumentowe	63
Kolejność operatorów	63
Powszechnie używane operatory	64

Dokonywanie logicznych wyborów za pomocą instrukcji warunkowych	65
Instrukcja if	65
Testowanie innych porównań	66
if..else	68
switch	69
Operator warunkowy	70
Powtarzanie operacji za pomocą pętli	71
Użycie pętli for	71
Używanie for..in oraz for each..in	73
Używanie while oraz do..while	74
Używać for czy while	75
Używanie break oraz continue	76
Komentowanie kodu	76
Typy komentarzy	77
Kiedy używać komentarzy	78
Podsumowanie	78
Rozdział 3. Programowanie z wykorzystaniem klas	81
Istota klas	81
Klasy mogą modelować rzeczywisty świat	82
Klasy zawierają dane i operacje	82
Klasy rozdzielają zadania	83
Klasy są typami	83
Klasy zawierają Twój program	83
Stosowana terminologia	84
Obiekt	84
Klasa	85
Instancja	85
Typ	85
Enkapsulacja	86
Zasada czarnej skrzynki	86
Enkapsulacja i polimorfizm	87
Pakiety	87
Unikalność klasy	88
Hierarchia	89
Kontrola widoczności	89
Kod dozwolony w pakietach	90
Używanie kodu z pakietów	91
Korzystanie z dziedziczenia	93
Struktura kodu zawierającego dziedziczenie	97
Dziedziczenie, typy oraz polimorfizm	98
Dziedziczenie a kompozycja	100
Zapobieganie dziedziczeniu	102
Używanie modyfikatorów dostępu do klas	104
Public i private	104
Protected	106
Internal	108
Przestrzenie nazw	109
Używanie metod w klasach	112
Konstruktory	113

Używanie właściwości w klasach	114
Metody dostępne	114
Unikanie efektów ubocznych	117
Przesłanianie zachowania	117
Używanie klasy bazowej	119
Używanie statycznych metod i właściwości	120
Zmienne statyczne	121
Stałe statyczne	123
Metody statyczne	125
Projektowanie interfejsów	127
Manipulowanie typami	133
Zgodność i koercja typów	133
Jawna konwersja typu	134
Określanie typów	136
Tworzenie klas dynamicznych	137
Podsumowanie	137
Rozdział 4. Praca z metodami i funkcjami	139
Funkcje	139
Różnice pomiędzy metodami i funkcjami	140
Wywołanie funkcji	140
Tworzenie własnych funkcji	141
Definiowanie funkcji	141
Przekazywanie parametrów do funkcji	142
Dostęp do obiektu arguments	145
Zwracanie wyników	146
Zwracanie wartości za pomocą wyrażenia return	147
Definiowanie funkcji za pomocą wyrażeń funkcyjnych	149
Dostęp do metod superklasy	150
Pisanie funkcji rekurencyjnych	151
Leczenie z rekursywności	152
Funkcje jako obiekty	153
Function a function	153
Metody i właściwości klasy Function	154
Podsumowanie	154
Rozdział 5. Walidacja programów	155
Wprowadzenie do błędów	155
Błędy kompilacji a błędy czasu wykonywania	156
Ostrzeżenia	156
Informacje zwrotne od programów Flash CS3 oraz Flex Builder	156
Naprawianie błędów	159
Powszechne typy błędów	160
Podsumowanie	162
Część II Praca z obiektami ActionScript 3.0	163
Rozdział 6. Używanie łańcuchów znaków	165
Praca z prostymi łańcuchami znaków	165
Konwertowanie obiektu łańcuchowego na typ prosty	166
Używanie sekwencji ucieczki	166

Konwersja na łańcuchy i z łańcuchów znakowych	167
Użycie toString	167
Rzutowanie na typ String	168
Konwersja łańcuchów znaków na inne typy	169
Łączenie łańcuchów znaków	169
Konwersja wielkości znaków w łańcuchu	170
Używanie poszczególnych znaków łańcucha	171
Uzyskanie liczby znaków w łańcuchu	171
Dostęp do poszczególnych znaków	171
Konwersja znaku na kod znaku	171
Wyszukiwanie w łańcuchu znaków	172
Wyszukiwanie przez podłańcuch	172
Wyszukiwanie za pomocą wyrażeń regularnych	173
Dzielenie łańcuchów	173
Podsumowanie	174
Rozdział 7. Praca z liczbami i funkcjami matematycznymi	175
Typy liczbowe	175
Zbiory liczb	175
Reprezentacja liczb	176
Cyfrowe reprezentacje liczb	177
Używanie liczb w ActionScript	179
Number	179
int	180
uint	180
Literały	181
Przypadki brzegowe	182
Manipulowanie liczbami	183
Konwersje liczbowe	183
Konwersje łańcuchów znaków	184
Obliczenia arytmetyczne	185
Obliczenia trygonometryczne	186
Generowanie losowości	187
Manipulowanie wartościami dat i czasów	188
Tworzenie daty	188
Czas epoki	190
Strefy czasowe	191
Uzyskiwanie dostępu do daty i jej modyfikacja	191
Arytmetyka dat	192
Czas wykonywania	193
Formatowanie daty	193
Podsumowanie	194
Rozdział 8. Używanie tablic	195
Podstawy tablic	195
Konstruktor Array	196
Tworzenie tablicy za pomocą literału tablicowego	197
Odwoływanie się do wartości w tablicy	197
Pobieranie liczby elementów w tablicy	198
Konwersja tablic na łańcuchy znaków	198

Dodawanie i usuwanie elementów z tablicy	199
Dołączenie wartości na końcu tablicy za pomocą concat()	199
Stosowanie operacji stosu — push() oraz pop()	200
Stosowanie operacji kolejki — shift() oraz unshift()	201
Cięcie i łączenie	201
Wstawianie i usuwanie wartości za pomocą splice()	202
Praca z podzbiorem tablicy przy użyciu slice()	202
Iteracje po elementach tablicy	203
Użycie pętli for	203
Użycie metody forEach()	203
Wyszukiwanie elementów	204
Zmiana kolejności w tablicy	204
Odwracanie kolejności tablicy za pomocą reverse()	205
Użycie funkcji sortujących	205
Przeprowadzanie operacji na wszystkich elementach tablicy	208
Przetwarzanie warunkowe za pomocą metod every(), some() oraz filter()	208
Uzyskiwanie wyników za pomocą metody map()	210
Alternatywne typy tablic	210
Praca z tablicami asocjacyjnymi	210
Używanie obiektu jako klucza wyszukiwania w słownikach	211
Używanie tablic wielowymiarowych	212
Podsumowanie	213
Rozdział 9. Używanie obiektów	215
Praca z obiektami	215
Klasy dynamiczne	215
Tworzenie obiektów	216
Dostęp do właściwości obiektu	217
toString()	217
Używanie instancji klasy Object jako tablicy asocjacyjnej	217
Porównywanie tablic, obiektów i słowników	218
Testowanie istnienia	220
Usuwanie właściwości	221
Iterowanie	221
Użycie obiektów dla nazwanych argumentów	222
Używanie obiektów jako zagnieżdżonych danych	223
XML jako obiekty	223
JSON	223
Podsumowanie	223
Rozdział 10. Praca z XML	225
Rozpoczęcie pracy z XML w ActionScript	225
Początki E4X	226
Praca z literałami XML	226
Krótkie wprowadzenie do operatorów i składni E4X	227
Klasy XML	228
Dostęp do wartości za pomocą E4X	229
Używanie operatora kropki do uzyskiwania dostępu do elementów	229
Użycie operatora @ do uzyskania dostępu do atrybutów	231
Dostęp do tekstu wewnątrz elementu	231
Operator dostępu do potomków	232

Dostęp do przodków	233
Iterowanie po dzieciach elementu	233
Filtrowanie wewnątrz XML	234
Konstruowanie obiektów XML	235
Łączenie węzłów XML	235
Usuwanie węzłów XML	238
Duplikowanie obiektu XML	239
Zamiana wartości w węzłach XML	240
Konwertowanie na łańcuchy znaków i odwrotnie	240
Konwertowanie łańcuchów znaków na XML	240
Konwertowanie XML na łańcuchy znaków	241
Wczytywanie danych XML z zewnętrznych źródeł	243
Gromadzenie metadanych węzłów XML	244
Używanie przestrzeni nazw	245
Praca z przestrzeniami nazw w ActionScript	246
Idąc jeszcze dalej	247
Praca z komentarzami i instrukcjami przetwarzania	248
Ustawianie opcji dla klasy XML	249
Podsumowanie	250
Rozdział 11. Praca z wyrażeniami regularnymi	251
Wprowadzenie do wyrażeń regularnych	251
Pisanie wyrażenia regularnego	252
Stosowanie wyrażeń regularnych	253
Metody łańcuchowe i metody klasy RegExp	253
Testowanie	253
Lokalizacja	254
Identyfikacja	256
Wyodrębnianie	257
Zastępowanie	259
Rozdzielanie	260
Konstruowanie wyrażeń	260
Zwykłe znaki	261
Znak kropki	261
Sekwencje ucieczki	261
Znaczenie metaznaków i metasekwencji	262
Klasy znaków	263
Kwantyfikatory	264
Kotwice i granice	265
Alternacja	267
Grupy	267
Flagi wyrażeń regularnych	268
Flaga global	269
Flaga ignoreCase	269
Flaga multiline	269
Flaga dotall	270
Flaga extended	271
Konstruowanie zaawansowanych wyrażeń	272
Dopasowywanie zachłanne i leniwe	272
Odwołania wsteczne	273

Grupy uprzedzone i nieprzechwycone	274
Grupy nazwane	275
Kwestie międzynarodowe	277
Używanie klasy RegExp	277
Budowanie dynamicznych wyrażeń za pomocą operacji łańcuchowych	277
Publiczne właściwości RegExp	278
Podsumowanie	279

Część III Praca z listą wyświetlania 281

Rozdział 12. Lista wyświetlania programu Flash Player 9 283

Lista wyświetlania	283
DisplayObject i DisplayObjectContainer	289
Czym jest klasa abstrakcyjna?	290
Pozycja x oraz y obiektu DisplayObject	290
Scena obiektu DisplayObject	290
Transformacja	291
Zmiana rozmiaru obiektu DisplayObject	292
Używanie trybu mieszania	292
Obiekt Graphics	293
Tworzenie wypełnień	294
Rysowanie linii w obiekcie graficznym	294
Rysowanie krzywych w obiekcie graficznym	295
Praca ze sceną	295
Użycie zdarzenia stageResize	295
Ustawianie wyrównywania sceny oraz trybu skalowania	296
InteractiveObject i SimpleButton	296
SimpleButton	297
Obiekty interaktywne dostępne za pomocą klawisza Tab	297
Zdarzenia aktywności i klawisza Tab	298
Właściwości myszy	299
Zdarzenia otrzymywane przez InteractiveObject	300
Shape	301
Tworzenie elementów interfejsu użytkownika za pomocą obiektów Sprite	301
Przeciąganie i upuszczanie	302
Użycie trybu buttonMode obiektu Sprite	303
Użycie hitArea	303
Użycie hitTestPoint	304
Zamiana kolejności potomków	304
Ponowne przypisywanie rodzica wyświetlanym obiektom	305
Praca z MovieClip	306
Użycie stop() oraz gotoAndPlay()	307
Właściwości totalFrames oraz framesLoaded	307
Przykłady zastosowania listy wyświetlania	308
Tworzenie procedury renderującej element	308
Tworzenie odbijającej się piłki	309
Sprawdzanie wystąpienia kolizji	310
Podsumowanie	312

Rozdział 13. Praca z obiektami DisplayObject w programie Flash CS3	315
Tworzenie symboli w programie Flash CS3	315
Ustawienie nazwy zmiennej Twojego symbolu na stole montażowym	318
Wykorzystanie własnej klasy dla własnego symbolu	319
Osadzanie w programie grafik bitmapowych	319
Osadzanie grafik w programie Flex	320
Dostęp do osadzonych klas graficznych	320
Podsumowanie	321
Rozdział 14. Drukowanie	323
Po co drukować z Flasha?	323
Sterowanie wydrukiem z Flasha	325
Wprowadzenie do klasy PrintJob	325
Uruchomienie żądania drukowania	326
Określanie celu drukowania oraz jego opcji formatujących	327
Potencjalne problemy podczas drukowania we Flashu	329
Dodawanie funkcji drukowania do aplikacji	330
Podsumowanie	334
Rozdział 15. Praca z tekstem i czcionkami	335
Pola tekstowe	336
Tworzenie nowego pola tekstowego	336
Dodanie tekstu do pola tekstowego	336
Ustawianie rozmiaru pola tekstowego	336
Ustawianie skali oraz obrotu pola tekstowego	337
Pobieranie łańcuchów znaków z pola tekstowego	338
Wyświetlanie HTML	338
Dodawanie obrazów lub plików SWF do pola tekstowego za pomocą 	340
Arkusze stylu dla pola tekstowego	341
Tworzenie tła dla pola tekstowego	341
Formatowanie tekstu	342
Tworzenie i użycie obiektu TextFormat dla pola tekstowego	342
Bardziej zaawansowana kontrola nad tekstem	349
Osadzanie czcionek i antialiasing	349
Właściwość gridFitType	350
Właściwości numLines oraz wordWrap	351
Ograniczenia znaków pola tekstowego	352
Właściwości przewijania	353
Wielowierszowość i wymiary tekstu	354
Wyświetlanie Unicode	356
Tworzenie wejściowych pól tekstowych	357
Uczynienie pola tekstowego wejściowym	357
Tabulatory dla wejściowych pól tekstowych	358
Nasłuchiwanie zdarzeń pola tekstowego	358
Zdarzenie textInput	358
Zdarzenia change i scroll	359
Zdarzenia focusIn oraz focusOut	360
Zdarzenia link	360
Podsumowanie	361

Część IV System obsługi zdarzeń 363**Rozdział 16. Działanie zdarzeń 365**

Podstawy działania zdarzeń	365
Zdarzenia sobotniego poranku	366
Terminologia zdarzeń	368
Klasa EventDispatcher	369
Używanie EventDispatcher	369
Używanie EventDispatcher poprzez kompozycję	373
Praca z obiektami Event	374
Tworzenie nowego zdarzenia	375
Dodawanie i usuwanie obiektów nasłuchujących zdarzeń	376
Usuwanie obiektu nasłuchującego z dyspozytora	377
Strumień zdarzenia	378
Fazy strumienia zdarzenia	378
Strumień zdarzenia w praktyce	381
Zapobieganie domyślnym zachowaniom	383
Podsumowanie	384

Rozdział 17. Praca ze zdarzeniami myszy i klawiatury 385

Podstawy zdarzeń MouseEvent	385
Współrzędne lokalne i sceny	388
Inne właściwości zdarzeń MouseEvent	390
Typy zdarzeń MouseEvent	390
MouseEvent.CLICK	391
MouseEvent.DOUBLE_CLICK	391
MouseEvent.MOUSE_DOWN	391
MouseEvent.MOUSE_MOVE	391
MouseEvent.MOUSE_OUT	391
MouseEvent.MOUSE_OVER	392
MouseEvent.MOUSE_UP	392
MouseEvent.MOUSE_WHEEL	392
MouseEvent.ROLL_OUT	392
MouseEvent.ROLL_OVER	392
FocusEvent.MOUSE_FOCUS_CHANGE	393
Używanie zdarzeń MouseEvent w połączeniu z użyciem myszy	394
Podstawy zdarzeń KeyboardEvent	395
Typy zdarzeń KeyboardEvent	396
FocusEvent.KEY_FOCUS_CHANGE	397
Zastosowanie właściwości keyCode	397
Zdarzenia IMEEvent	398
Podsumowanie	399

Rozdział 18. Korzystanie z czasomierzy 401

Podstawy pracy z czasomierzem	402
Tworzenie czasomierza	402
Nasłuchiwanie zdarzeń czasomierza	403
Uruchamianie, zatrzymywanie i resetowanie czasomierza	403
Obsługa zdarzenia TimerEvent	404
Uzyskanie referencji do czasomierza	405
Opóźnianie wykonania funkcji	406

Tworzenie zegara światowego	407
Bonus	408
Starsze funkcje czasomierza	408
Użycie getTimer()	409
Podsumowanie	409

Część V Obsługa błędów 411

Rozdział 19. Czym są błędy 413

Drogi wystąpienia błędu	413
Korzystanie z wyjątków	414
Rzucanie wyjątków	415
Przechwytywanie wyjątków	415
Przepływ wyjątku	417
Wyjątki nieprzechwycone	419
finally	420
Ponowne rzucanie wyjątków	421
Przechwytywanie błędów generowanych przez Flasha	421
Własne wyjątki	423
Obsługa błędów asynchronicznych	424
Podsumowanie	425

Rozdział 20. Korzystanie z debugera AVM2 427

Wprowadzenie do debugowania	427
Uruchamianie debugera	428
Uruchamianie i zatrzymywanie debugera Flash CS3	429
Uruchamianie i zatrzymywanie debugera Flex Builder 3	430
Porównanie debuggerów	432
Przejmowanie sterowania nad wykonywaniem	433
Zatrzymanie przy nieprzechwyconym wyjątku	433
Zatrzymanie w punkcie przerwania	434
Zatrzymanie na żądanie	435
Odsłaniając kurtynę	436
Interpretacja panelu Zmienne	437
Panel Variables w programie Flex Builder i wyrażenia czujki	438
Nawigowanie po kodzie	439
Kontynuuj	439
Stos wywołań	440
Wejdz (step into)	441
Wykonaj krokowo (step over)	442
Wyjdz lub return	442
Debugowanie prostego przykładu	443
Efektywne wykorzystanie debugera	445
Podsumowanie	446

Rozdział 21. Aplikacja tolerancyjna na błędy 449

Tworzenie strategii	449
Określanie błędów do obsługi	450
Określanie kategorii błędu	451
Logi błędów	452
Informowanie użytkownika	454

Degradacja stylów — przykład	455
Podsumowanie	457

Część VI Praca z danymi zewnętrznymi 459

Rozdział 22. Podstawy pracy w sieci 461

URLRequest	462
navigateToURL()	463
GET i POST	463
URLLoader	465
Podstawy URLLoader	465
Dalsze wykorzystanie URLLoader	468
URLVariables	469
URLStream	470
sendToURL()	471
Użycie obiektu Loader	471
Bezpieczeństwo w programie Flash Player	474
Podsumowanie	476

Rozdział 23. Komunikacja z technologiami serwerowymi 479

Komunikacja poprzez URLLoader	479
Wykorzystanie PHP w komunikacji z Flashem	480
Wysyłanie danych bez ich wczytywania	481
XMLSocket	481
Tworzenie obiektu XMLSocket	482
Strona serwerowa XMLSocket	482
Flash Remoting	483
NetConnection	484
Responder	485
Rozwiązania serwerowe Flash Remoting	486
Podsumowanie	488

Rozdział 24. Zapisywanie danych na lokalnym komputerze za pomocą SharedObject 489

Porównanie sposobów trwałego przechowywania	489
Przechowywanie informacji w lokalnym obiekcie współdzielonym	490
Przechowywanie informacji na serwerze	490
Przechowywanie informacji w plikach cookie przeglądarki	490
Sytuacje odpowiednie do zastosowania obiektów współdzielonych	491
Użycie obiektów SharedObject	492
Uzyskiwanie obiektu SharedObject	492
Odczyt i zapis w SharedObject	493
Usuwanie informacji z SharedObject	493
Zapisywanie informacji	494
Udostępnianie danych pomiędzy plikami SWF	494
Żądanie bezpiecznego połączenia	495
Udostępnianie obiektów współdzielonych plikom SWF ActionScript 1.0 oraz 2.0	496
Praca z ograniczeniami rozmiaru	497
Ustawienia użytkownika	497
Żądania użytkownika	498
Pytanie o miejsce zanim będzie za późno	499

Wcześniejsze pytanie o miejsce	499
Korzystanie z flush()	500
Wyświetlanie wykorzystywanego miejsca	501
Przechowywanie własnych klas	501
Zapisywanie własnych klas bez modyfikacji	501
Tworzenie klas samoserializujących	503
Użycie koncepcji serializacji dla wywołań zwrotnych	505
Podsumowanie	505

Rozdział 25. Zarządzanie wysyłaniem i pobieraniem plików 507

FileReference	507
Wysyłanie plików	508
Wybór pliku do wysłania	508
Określanie, czy plik został wybrany	509
Pobieranie właściwości pliku	510
Wysyłanie pliku	512
Dodanie do aplikacji możliwości wysyłania plików	513
Pobieranie pliku	515
Podsumowanie	516

Część VII Wzbogacanie programów dźwiękiem i filmem 517

Rozdział 26. Praca z dźwiękiem 519

Jak działa dźwięk w AS3	519
Poznanie klas dźwiękowych AS3	519
Praca ze zdarzeniami dźwiękowymi	520
Tworzenie obiektu Sound	521
Wczytywanie pliku z zewnętrznego pliku lub URL	521
Osadzanie dźwięków w programie	523
Kontrola odtwarzania dźwięku	524
Odtwarzanie i zatrzymywanie dźwięku	524
Ustawianie punktu początkowego oraz liczby pętli dźwiękowych	525
Przebijanie do przodu, do tyłu, wstrzymywanie i ponowne rozpoczynanie odtwarzania	525
Stosowanie transformacji dźwiękowych	528
Zmiana głośności i balansu dźwięku	528
Praca z metadanymi dźwięku	529
Sprawdzanie rozmiaru pliku dźwiękowego	529
Uzyskiwanie danych ID3 utworu	529
Obliczanie danych spektrum	530
Wykrywanie możliwości dźwiękowych	532
Podsumowanie	533

Rozdział 27. Dodawanie wideo 535

Praca z plikami Flash Video	535
Tworzenie pliku FLV za pomocą aplikacji Flash Video Encoder	535
Użycie RTMP do uzyskiwania dostępu do plików FLV	536
Użycie HTTP do uzyskania dostępu do plików FLV	537
Klasa Video oraz NetStream	538
Wczytywanie plików FLV do filmu Flash	540
Utworzenie połączenia HTTP prowadzącego do pliku FLV	541
Wyświetlanie danych NetStream w obiekcie Video	541

Sprawdzanie informacji statusu klasy NetStream	541
Uzyskiwanie metadanych	542
Tworzenie aplikacji odtwarzającej pliki FLV	543
Podsumowanie	548
Rozdział 28. Dostęp do mikrofonów i kamer	549
Kamera	549
Podłączanie kamery do obiektu Video	551
Mikrofon	552
Praca z mikrofonem	553
Serwery multimedialne	555
Podsumowanie	556
Część VIII Programowanie grafiki i ruchu	557
Rozdział 29. Stosowanie filtrów do grafik	559
Podstawy filtrów	560
Stosowanie filtrów	561
Filtry rozmycia	561
Dodawanie filtru cienia	562
Filtry fazy	563
Filtry blasku	565
Dodawanie filtru fazy gradientowej	566
Filtr macierzy kolorów	568
Dodawanie filtru konwolucyjnego	574
Filtr blasku gradientowego	575
Filtr mapy przemieszczeń	575
Dodawanie więcej niż jednego filtru	576
Obracanie obiektów z filtrami	577
Podsumowanie	579
Rozdział 30. Programowe rysowanie grafik wektorowych	581
Linie i style linii	581
Ustawianie stylu linii	582
Przesuwanie pióra bez rysowania	585
Rysowanie prostej linii	585
Rysowanie krzywej	586
Dodanie prostego, jednokolorowego wypełnienia	587
Dodawanie wypełnienia bitmapowego	588
Praca z gradientami	590
Czyszczenie wcześniej narysowanej grafiki	594
Tworzenie kształtów	594
Rysowanie okręgów	595
Rysowanie elips	596
Rysowanie zaokrąglonych prostokątów	596
Wypełnianie kształtów	597
Maski	599
Zaawansowane maski	600
Tworzenie aplikacji rysującej	601
Podsumowanie	607

Rozdział 31. Programowanie animacji	609
Flash Player a animacja	609
Szybkość odtwarzania	609
Działanie programu Flash Player	610
Animowanie za pomocą samego ActionScript	612
Animowanie za pomocą czasu	612
Animowanie za pomocą klatek	613
Animacja i prędkość	614
Animowanie w programie Flash	615
Przegląd: animacje automatyczne, klatki kluczowe i zmiana dynamiki	616
XML ruchu	617
Użycie pakietu ruchu Flasha	625
Animowanie za pomocą środowiska Flex	627
Zestawy animacyjne firm trzecich	627
Pakiet ruchu Flash	627
Tweeners	628
AnimationPackage	628
Podsumowanie	629
Rozdział 32. Transformacje grafiki	631
Praca z transformacjami macierzowymi	631
Wykorzystanie transformacji macierzowych	635
Praca z transformacjami kolorów	637
Stosowanie transformacji kolorów	638
Pobieranie i ustawianie koloru	638
Użycie tint dla obiektów wyświetlanych	639
Przywracanie kolorów	640
Transformacja kolorów	640
Podsumowanie	641
Rozdział 33. Programowe rysowanie grafik bitmapowych	643
Tworzenie obiektu BitmapData	643
Metoda wykorzystująca konstruktor	644
Tworzenie instancji osadzonych zasobów	644
Wyświetlanie obrazów BitmapData	646
Praca z właściwościami BitmapData	647
Kopiowanie obrazów	647
Kopiowanie z obiektów wyświetlanych	648
Wczytywanie obrazów BitmapData	651
Kopiowanie z obiektów BitmapData	653
Stosowanie transformacji kolorów	661
Stosowanie wypełnień	662
Stosowanie wypełnień prostokątnych	662
Stosowanie wypełnień zalewanych	662
Wyznaczanie obszarów według koloru	663
Stosowanie efektów	664
Zastępowanie kolorów z wykorzystaniem progu	664
Używanie przenikania pikseli	666
Ponowne mapowanie palety kolorów	669

Tworzenie szumu	670
Dodawanie szumu	670
Dodawanie szumu Perlina	673
Stosowanie filtrów	676
Podsumowanie	678

Część IX Praca z danymi binarnymi 679

Rozdział 34. Praca z danymi binarnymi	681
Tworzenie tablicy bajtów	681
Zapis do tablicy bajtów	682
Odczyt z tablicy bajtów	682
Powszechnie zastosowania tablic bajtów	683
Wyznaczanie spektrum dźwiękowego	683
Wczytywanie obrazów	684
Kopiowanie obiektów	685
Własna serializacja danych	686
Praca z binarnymi gniazdami	688
Podsumowanie	691

Część X Wdrażanie programu 693

Rozdział 35. Wdrażanie zawartości Flash	695
Osadzanie treści Flash na stronie internetowej	695
Osadzanie treści Flash za pomocą SWFObject	697
Ustawianie opcji programu Flash Player	698
Przezroczysty Flash	699
Pełnoekranowy Flash	699
Przekazywanie zmiennych do pliku SWF	700
Automatyczne uaktualnianie programu Flash Player	701
Podsumowanie	702
Rozdział 36. Komunikacja z JavaScript	703
Komunikacja pomiędzy JavaScript a Flashem	703
Klasa ExternalInterface	703
Podsumowanie	707

Rozdział 37. Wykorzystanie połączeń lokalnych do komunikacji pomiędzy filmami Flash	709
Tworzenie aplikacji wysyłającej	709
Wysyłanie parametrów	710
Sprawdzanie statusu wysyłania	710
Tworzenie aplikacji odbierającej	710
Wysyłanie i odbieranie pomiędzy domenami	711
Aplikacja wysyłająca	712
Aplikacja odbierająca	712
Podsumowanie	713

Skorowidz	715
------------------------	------------

Rozdział 4.

Praca z metodami i funkcjami

W tym rozdziale:

- ◆ Ponowne wykorzystanie bloków kodu poprzez definiowanie własnych funkcji
- ◆ Wywoływanie funkcji i przekazywanie argumentów za pomocą operatora wywołania
- ◆ Otrzymywanie wyników poprzez zwracanie wartości z funkcji
- ◆ Praca z funkcjami jako obiektami
- ◆ Tworzenie funkcji rekurencyjnych

Teraz, gdy wiesz już wszystko na temat zmiennych, zapewne będziesz chciał coś z nimi zrobić. W tym miejscu do akcji wkraczają metody i funkcje. Funkcje są blokami kodu wielokrotnego zastosowania, które mogą być definiowane w Twoich własnych klasach i często są stosowane w całym API ActionScript 3.0.

Funkcje umożliwiają organizowanie kodu w niezależne fragmenty. Mogą być używane do zwracania różnych wyników w zależności od dostarczonych im danych. Być może najważniejsze jest jednak to, że funkcje mogą enkapsulować wewnątrz klasy zachowania i funkcjonalność oraz oferować publiczny interfejs dla tej funkcjonalności. W niniejszym rozdziale opisano sposoby wykorzystywania funkcji i tworzenia od podstaw własnych funkcji.

Jedną z rzeczy, jakie należy zapamiętać w związku z ActionScript, jest to, że *każda zmienna i część klasy jest obiektem*. Funkcje nie są tutaj wyjątkiem. Choć może być to trudne do wyobrażenia, funkcje są instancjami klasy `Function` i zawierają własne metody oraz właściwości. W tym rozdziale opiszemy, jak używać funkcji jako obiektów.

Funkcje

Mówiąc najprościej, funkcja jest fragmentem kodu zachowanym w postaci zapisanej procedury, która może być uruchomiona w razie potrzeby poprzez wpisanie nazwy funkcji lub wywołanie funkcji. Każdy napisany przez Ciebie program będzie w dużym stopniu bazował na funkcjach.

Różnice pomiędzy metodami i funkcjami

Będziesz spotkać się ze słowami „metoda” oraz „funkcja” (a czasami nawet z określeniem „domknięcie funkcji”) oznaczającymi blok kodu nadający się do wielokrotnego zastosowania. Pomimo tego iż często nazw tych używa się zamiennie, występuje pomiędzy nimi niewielka różnica. Metoda jest dowolną funkcją zdefiniowaną wewnątrz klasy. Metody powinny być logicznie powiązane z instancją klasy i dążyć do spójności z obiektem, w którym są zdefiniowane. Np. klasa `Kitty` może definiować funkcję o nazwie `meow()`. W tym przypadku `meow()` będzie metodą klasy `Kitty`.

Ogólnie pojęcia „metoda” oraz „funkcja” różnią się więc tylko nazwami — innymi słowy, metody nie posiadają żadnych dodatkowych możliwości, nie występuje też słowo kluczowe „method”. Oto przykład wywołania wymyślonej funkcji:

```
addElementToGroup(myGroup, myElement);
```

A teraz wywołanie tej samej funkcji, gdy napisana będzie jako metoda:

```
myGroup.addElement(myElement);
```

Jak widzisz, powyższe wywołania są do siebie podobne, ale drugie zostało tak zaprojektowane, aby opierać na instancji `myGroup`.

W tej książce używamy obu terminów. Używamy słowa „funkcja” dla zastosowania ogólnego, a podczas opisywania funkcjonalności konkretnej funkcji zdefiniowanej wewnątrz klasy używamy pojęcia „metoda”. Ponieważ prawie wszystkie funkcje w ActionScript znajdują się w klasach, większość z nich jest metodami.



Inaczej niż w AS2, który czasami wykorzystuje klasę `Delegate`, w AS3 metody są powiązane z instancjami, w których się znajdują. Wykonają się tak, jakby zostały wywołane z instancji, nawet wtedy gdy funkcja zostanie przekazana do innego zasięgu. Ta właściwość ActionScript 3.0 znana jest jako **domknięcie metody**.

Wywołanie funkcji

Wykonanie kodu funkcji lub metody znane jest jako **wywołanie** lub **uruchamianie**. Funkcje w ActionScript są wywoływane za pomocą nazwy funkcji, po której występują nawiasy `()`. Oficjalnie nawiasy te nazywane są **operatorem wywołania**.

Dodatkowe informacje mogą być przekazywane do Twoich funkcji poprzez dodanie **argumentów** (znanych także jako **parametry**) umieszczanych między nawiasami i rozdzielanych przecinkami. Niektóre funkcje będą posiadać argumenty opcjonalne lub wymagane. Liczba i typ wartości przekazywanych do funkcji musi odpowiadać definicji funkcji, zwanej także **sygnaturą metody**. Jeszcze inne funkcje nie wymagają wcale argumentów, a w ich wywołaniu używa się pustej pary nawiasów.

Poniżej widzisz dwa poprawne wywołania funkcji:

```
trace("Witaj świecie!");  
addChild(mySprite);
```



Nie zapomnij o dodaniu nawiasów podczas wywoływania funkcji. W przeciwnym razie wyznaczona zostanie wartość funkcji jako zmiennej (obiektu typu `Function`) zamiast wykonania się kodu wewnątrz funkcji, co doprowadzi do nieoczekiwanych wyników. W dalszej części tego rozdziału dowiesz się, jak i kiedy używać funkcji bez operatora wywołania.

Wywoływanie funkcji a wywoływanie metod

Aby uzyskać dostęp do metod wewnątrz konkretnej instancji, użyj nazwy obiektu zawierającego metody, a po niej umieść kropkę i wywołanie metody. Taki sposób wywołania nazywa się **notacją kropkową**. Możesz traktować to jak wywołanie funkcji w kontekście obiektu.

```
var nameList:Array = new Array();
nameList.push("Daniel", "Julia", "Paweł");
nameList.reverse();
trace(nameList); // Wyświetli: "Paweł", "Julia", "Daniel"
```



Niektóre instrukcje ActionScript lub operatory, jak `typeof` czy `delete`, nie są technicznie funkcjami, nawet jeśli tak się zachowują. Z tego też powodu nie będziesz musiał dodawać nawiasów wokół argumentów dla tych poleceń i nie będziesz w stanie uzyskać dostępu do metod i właściwości klasy `Function` dla tych zarezerwowanych słów. Operatory te, w przeciwieństwie do większości funkcji, istnieją globalnie i będą dostępne w dowolnym miejscu kodu.

Tworzenie własnych funkcji

Aby utworzyć własne metody, należy dodać funkcje do swoich plików przechowujących deklaracje klas. Nazywa się to **definiowaniem** lub **deklarowaniem** funkcji. Przyjrzyjmy się tej operacji.

Definiowanie funkcji

Funkcje posiadają następującą strukturę:

```
public function doSomething(arg1:Object, arg2:Object):void {
    // wykonywany kod znajduje się tutaj
}
```

Rozbijmy to na poszczególne człony:

- ♦ `public` — modyfikator dostępu będący zazwyczaj słowem kluczowym, takim jak `public`, `private` czy `internal`, używany jest do określania dostępu do tej metody dla innych klas. Jeśli definiujesz funkcję poza pakietem (np. na osi czasu w programie Flash CS3), powinieneś pominąć tę część. Zawsze powinieneś definiować przestrzeń nazw dla swoich metod. W celu uproszczenia niektóre przykłady w tej książce mogą pomijać tę część.
- ♦ `function` — następnie występuje słowo kluczowe `function`. Jest ono zawsze wymagane podczas definiowania funkcji, podobnie jak słowo `var` wymagane jest podczas definiowania zmiennej.

- ♦ `doSomething` — bezpośrednio po słowie kluczowym `function` występuje nazwa funkcji. Jest ona zarazem poleceniem, jakie wywołasz, gdy zechcesz wykonać daną funkcję.



Najlepsze są opisowe nazwy funkcji. Opisują akcję i mogą być czytane z łatwością, jakbyś czytał zdanie. Niepisana zasada mówi, że nazwa funkcji powinna zaczynać się od czasownika. Np. nazwa `button()` nie jest tak wyrazista jak `drawButton()`. Idąc tym tropem, najlepszymi nazwami dla zmiennych są rzeczowniki. Możesz więc łączyć czasownik z rzeczownikami i tworzyć krótkie zdania, jak `snakeHandler.feed(python, mouse)`.

- ♦ `(arg1:Object, arg2: Object)` — za nazwą funkcji występuje rozdzielona przecinkami lista argumentów, znajdująca się pomiędzy parą nawiasów. Po każdym argumencie powinien wystąpić dwukropek oraz typ danych tego argumentu.
- ♦ `:void` — za nawiasami występuje kolejny dwukropek i typ danych dla wartości zwracanej przez funkcję. Jest to wartość, jaka zostanie zwrócona przez funkcję po zakończeniu jej wykonywania. W tym przypadku zwracany typ to `void`, ponieważ funkcja nie posiada instrukcji zwracającej. Więcej na temat typów zwracanych dowiesz się w dalszej części tego rozdziału.
- ♦ `{...}` — cały kod wykonywany po wywołaniu funkcji znajduje się pomiędzy dwoma nawiasami klamrowymi `{}`. Nazywa się go ciałem funkcji.

Wszystkie funkcje wymagają słowa kluczowego `function`, nazwy funkcji, nawiasów i ciała funkcji. Pozostałe obszary nie są wymagane, co nie oznacza, że nie powinieneś ich używać.

Przekazywanie parametrów do funkcji

Funkcje stają się o wiele bardziej interesujące i bardziej przydatne, gdy dostarczysz im jakichś danych zewnętrznych. Można to osiągnąć poprzez dodanie do Twojej definicji funkcji *argumentów*, nazywanych również *parametrami*. Aby tego dokonać, po prostu umieść jeden lub więcej argumentów w nawiasach w instrukcji funkcji. Nazwy, jakie tutaj zdefiniujesz, będą dostępne w trakcie wykonywania jako zmienne lokalne i będziesz mógł ich używać podczas wykonywania kodu.

Nie wszystkie funkcje będą wymagać parametrów. Niektóre będą wywoływane z pustymi nawiasami.

Przyjrzyj się przykładowi wyznaczającemu obwód okręgu, w którym jako parametr przekazywana jest średnica okręgu:

```
function getCircumference(diameter:Number):Number {
    return Math.PI * diameter;
}
```

Jak widzisz, parametry te mogą być ponownie zapisywane i istnieją w lokalnym zasięgu funkcji. Oznacza to, że przesłaniają zmienne posiadające te same nazwy i istniejące w zasięgu klasy lub zasięgu globalnym.



W trybie standardów liczba argumentów musi odpowiadać definicji funkcji. Różni się to od podejścia stosowanego w poprzednich wersjach ActionScript, gdzie symetria pomiędzy argumentami w definicji funkcji oraz wywołaniem funkcji nie była wymagana.

Przekazywanie przez referencję lub przez wartość

W kontekście przekazywania wartości do funkcji ActionScript 3.0 traktuje proste oraz złożone typy danych w odmienny sposób. Proste typy danych są przekazywane *przez wartość* — czyli ich wartość jest kopiowana do funkcji, pozostawiając oryginał niezmieniony, niezależnie od tego, co wydarzy się wewnątrz funkcji. Złożone typy danych, jak Array, są przekazywane *przez referencję*, co powoduje przekazanie oryginalnego obiektu zamiast duplikatu. Reguły te stosuje się w ten sam sposób podczas przypisywania wartości zmiennym.

Jeżeli używasz komputera, a jestem pewien, że większość z Czytelników tej książki go używa, na pewno znasz różnicę pomiędzy kopiowaniem plików a tworzeniem skrótów (lub aliasów w Mac OS). Zgodnie z tym, co pokazano na rysunku 4.1, przekazywanie poprzez wartość jest bardzo podobne do powielenia pliku, ponieważ oryginalny plik pozostaje bez zmian na swoim miejscu. Przekazywanie przez referencję przypomina tworzenie skrótu do oryginalnej wartości. Gdy utworzysz skrót do pliku tekstowego, otworzysz skrót i edytujesz plik, wprowadzone przez Ciebie zmiany zostaną zapisane w oryginalnym pliku, do którego się podłączyłeś.

Rysunek 4.1.

Kopiowanie pliku jest podobne do przekazywania parametru przez wartość, podczas gdy tworzenie skrótu przypomina przekazanie parametru przez referencję



Poniżej przedstawiony został przykład przekazywania parametru przez wartość. Zauważ, że oryginalna wartość nie ulega zmianie:

```
function limitPercentage(percentage:Number):Number {
    // upewnij się, czy wartość procentowa jest mniejsza od 100
    percentage = Math.min(percentage, 100);
    // upewnij się, czy wartość procentowa jest większa od 0
    percentage = Math.max(percentage, 0);
    return percentage;
}
var effort:Number = 110;
var trimmedPercentage:Number = limitPercentage(effort);
trace(effort, "%"); // wyświetli: 110%
trace(trimmedPercentage, "%"); // wyświetli: 100%
```

Początkowa wartość zmiennej `effort` nie zmieniła się, chociaż zmienna argumentu `percentage` zmieniła wartość podczas wykonywania funkcji.

Dla typów złożonych sprawa wygląda inaczej. Są przekazywane *przez referencję*, co oznacza, że argument zachowuje się jak skrót lub odsyłacz do oryginalnego pliku. Zmiany argumentu są odzwierciedlane jako zmiany bezpośrednio na przekazywanej wartości. Większość typów danych przekazuje referencję podczas przypisywania zmiennej wartości. Poniższy kod przedstawia, jak zmienna `employee` jest bezpośrednio łączona z obiektem przekazywanym jako parametr.

```
function capitalizeEmployeeName(employee:Object):void {
    if (employee.name != null) {
        employee.name = employee.name.toUpperCase();
    }
}
var person:Object = {name: "Jan Kowalski"};
capitalizeEmployeeName(person);
trace(person.name); // wyświetli: JAN KOWALSKI
```

Jak widać w tym przykładzie, zmiana wielkości liter jest przeprowadzana bezpośrednio na zmiennej `employee`. Dzieje się tak dlatego, że wartość `employee` odwołuje się bezpośrednio do oryginalnej kopii `person` — dlatego nazwisko osoby jest również powiązane.

Poniżej wypisane typy danych przekazywane są przez wartość:

```
String
Number
int
uint
Boolean
Null
void
```

Pozostałe typy danych są przekazywane przez referencję.

Ustawianie wartości domyślnych

Nowością w ActionScript 3.0 jest możliwość ustawiania domyślnych wartości dla argumentów metody. Aby tego dokonać, dodaj za nazwą argumentu znak równości (=) oraz domyślną wartość:

```
function showGreeting(name:String = "obcy"):void {
    trace("Witaj, " + name + ", miło Cię poznać.");
}
showGreeting("Andrzej"); // wyświetli: Witaj, Andrzej, miło Cię poznać.
showGreeting(); // wyświetli: Witaj, obcy, miło Cię poznać.
```

Jak widzisz na przykładzie drugiego wywołania, imię zostało zastąpione domyślną wartością "obcy".

Podczas stosowania domyślnych wartości należy pamiętać o jeszcze jednej zasadzie. Wszystkie argumenty posiadające domyślną wartość są uznawane za opcjonalne. Jako takie muszą być umieszczone na końcu listy argumentów. Dlatego też następujący kod:

```
function storeAddress(name:String, zip:String = null, email:String)
```

nie jest poprawnym kodem, ponieważ `email` jest parametrem wymaganym, a występuje za `zip`, który jest opcjonalny. Poprawna kolejność wygląda w tym przypadku następująco:

```
function storeAddress(name:String, email:String, zip:String = null)
```

Użycie argumentu reszty (...)

ActionScript 3.0 wprowadza nowe rozwiązanie nazywane **parametrem reszty** (...). Parametr reszty oznaczany jest symbolem `...`, reprezentującym zmienną liczbę argumentów w postaci tablicy. Po nim następuje nazwa tej tablicy. Pisany jest jako trzy kropki, po których następuje nazwa używana dla tablicy zawierającej te wartości. Dodanie do własnej funkcji parametru reszty umożliwi Ci wywoływanie funkcji z dowolną liczbą parametrów. Jeśli np. potrzebujesz funkcji dodającej do siebie dowolną ilość liczb, możesz użyć parametru reszty:

```
function sum(... numbers):Number {  
    var result:Number = 0;  
    for each (var num:Number in numbers) {  
        result += num;  
    }  
    return result;  
}  
trace(sum(1,2,3,4,5)); // wyświetli: 15
```

Wartości przekazane do funkcji zawarte są w tablicy reszty o nazwie `numbers`. Dostęp do elementów tej tablicy uzyskasz, przechodząc w pętli po jej kolejnych wartościach.



Tablice zostały dokładniej omówione w rozdziale 8.

Parametr reszty może zostać również użyty z innymi, wymaganymi parametrami. Wymagane parametry jak zwykle będą posiadać swoje własne nazwy i będą istnieć niezależnie od tablicy reszty. Wszystkie dodatkowe parametry po tych wymaganych zapisane zostaną w tablicy reszty. Zmodyfikujemy poprzedni przykład tak, aby wymagał przekazania przynajmniej jednego argumentu:

```
function sum(base:Number, ... numbers):Number {  
    var result:Number = base;  
    for each (var num:Number in numbers) {  
        result += num;  
    }  
    return result;  
}  
trace(sum()); // zgłosi błąd czasu wykonywania  
trace(sum(1,2,3,4,5)); // wyświetli: 15
```

Dostęp do obiektu arguments

Klasa `Function` definiuje pojedynczą właściwość, która jest wykorzystywana do przechowywania informacji na temat danych przekazywanych do funkcji podczas jej wywoływania. Właściwość ta nazywa się `arguments`. Jest to obiekt specjalnego typu danych `Arguments`, który pomimo iż zachowuje się podobnie do tablicy, technicznie nie jest tablicą. Każda funkcja jest instancją klasy `Function`, przez co posiada właściwość `arguments`.



Właściwość `arguments` w ActionScript 3.0 straciła na znaczeniu i pozostawiono ją jedynie w celu obsługi starszego kodu. Zalecamy zamiast niej używanie nowego parametru reszty (...).

Parametr reszty i obiekt `arguments` nie mogą być używane razem. Dołączenie argumentu reszty w Twojej funkcji wyłączy obiekt `arguments`.

Użyteczność obiektu `arguments` jest blokowana w trybie standardów, ponieważ tryb ten wymaga dostarczenia dokładnej liczby argumentów zdefiniowanych w deklaracji funkcji. Jeśli zdecydujesz się na użycie obiektu `arguments`, możesz wyłączyć kompilowanie w trybie standardów.

Uzyskiwanie wartości z właściwości `arguments`

Właściwość `arguments` jest bardzo podobna do parametru reszty w tym, że zawiera każdy parametr, który jest przekazywany do funkcji w postaci uporządkowanej tablicy. Główna różnica pomiędzy nimi polega na tym, że w tablicy argumentów znajdują się wszystkie nazwane, jak również nienazwane argumenty, podczas gdy w parametrze reszty zawarte są tylko nienazwane argumenty.

Do parametrów przekazywanych do funkcji można uzyskać dostęp za pomocą składni tablicowej. Pierwszy przekazany parametr będzie posiadać indeks 0, drugi indeks 1 i tak dalej. Poniższy przykład zostanie skompilowany poprawnie, gdy tryb standardów będzie wyłączony:

```
function sum():Number {  
    var result:Number = 0;  
    for each (var num:Number in arguments) {  
        result += num;  
    }  
    return result;  
}  
trace(sum(1,2,3,4,5)); // wyświetli: 15
```



Właściwość `arguments.caller` nie jest już dostępna w ActionScript 3.0.

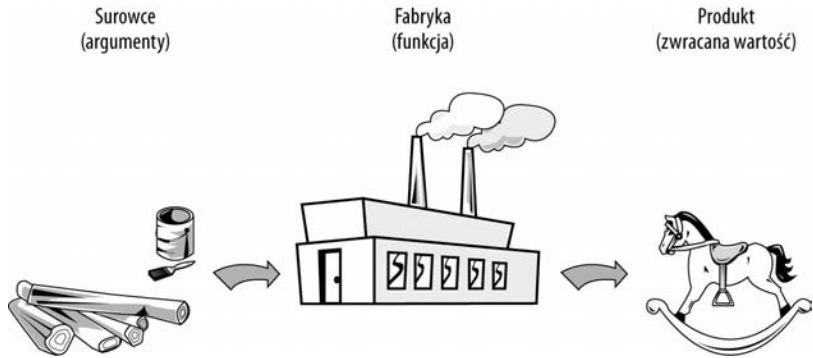
Zwracanie wyników

Funkcje potrafią o wiele więcej niż zwykle wykonywanie predefiniowanego kodu — to zaledwie połowa ich zastosowania. Funkcje w ActionScript potrafią podobnie jak funkcje matematyczne obliczać wartości w oparciu o wprowadzone zmienne. Możliwość ta jest bardzo pożyteczna.

Możesz traktować funkcję tak, jakby była fabryką przekształcającą surowce w gotowe produkty, co przedstawiono na rysunku 4.2. W tym przypadku surowce są argumentami przekazywanymi do wywołania funkcji, a końcowy produkt jest wartością zwracaną przez funkcję.

Rysunek 4.2.

Funkcje, podobnie jak fabryki, przekształcają surowce w gotowy produkt



Zwracanie wartości za pomocą wyrażenia return

W najprostszej formie instrukcja zwracająca jest słowem kluczowym `return`, po którym następuje zwracana wartość. Wartość może być typu prostego, może być zmienną, wynikiem działania innej funkcji czy dowolnym innym typem wyrażenia.

```
return someValue;
```

Zauważ, że w instrukcji `return` nie używa się nawiasów. Wynika to z faktu, że nie jest to funkcja, ale raczej operator. Możesz natknąć się czasem na nawiasy — np. `return (someValue)` — ale nie są one wymagane.

W większości sytuacji będziesz chciał w określony sposób manipulować argumentami wewnątrz funkcji, a następnie zwrócić nową, obliczoną wartość. Poniższy przykład oblicza obwód okręgu poprzez pobranie średnicy jako parametru `diameter`, pomnożenie jej przez matematyczną stałą `pi` i zwrócenie wyniku w postaci liczby:

```
function getCircumference(diameter:Number):Number {
    var circumference:Number = diameter * Math.PI;
    return circumference;
}
trace(getCircumference(25)); // wyświetli: 78.53981633974483
```

Jak widzisz, wywołanie metody `getCircumference()` może zostać użyte w instrukcji `trace`, tak jakby było zmienną. Wyniki działania funkcji definiującej typ zwracany mogą być również zapisywane w zmiennej. Realizuje się to poprzez umieszczenie zmiennej z lewej strony znaku równości.

Gdy tylko w funkcji wykonana zostanie instrukcja `return`, funkcja kończy się, a reszta jej kodu nie jest przetwarzana. Z tego też powodu każda funkcja może zwrócić tylko jedną wartość. Instrukcja `return` może też zostać użyta do przedwczesnego zakończenia funkcji.

```
function getColor():String {
    var color:String = "czerwony";
    return color;
    color = "niebieski";
    return color;
}
var color:String = getColor();
trace(color); // wyświetli: czerwony
```

Ta metoda zawsze będzie zwracać „czerwony”, ponieważ wykonywanie kodu tej funkcji zostaje zatrzymane po pierwszej instrukcji zwracającej.



Chcąc zwrócić z funkcji więcej niż jedną wartość, możesz utworzyć obiekt lub jeszcze lepiej własną klasę, która będzie przechowywać wszystkie Twoje odpowiedzi w jednej zmiennej.

```
return {firstName:"Paul", lastName:"Newman"};
```

Definiowanie typu zwracanej wartości dla Twojej funkcji

Gdy definiujesz typ danych dla zmiennej, stałej czy argumentu za pomocą dwukropka (:) po nazwie obiektu, wskazuje to kompilatorowi, jaki rodzaj informacji powinien być przechowywany w tym obiekcie. Funkcje działają w ten sam sposób.

Wszystkie funkcje definiujące **zwracany typ** muszą honorować go poprzez zwracanie wartości za pomocą instrukcji `return`, odpowiadającej typowi danych. I odwrotnie: wszystkie funkcje używające instrukcji `return` powinny posiadać typ zwracany. Zwracany typ — podobnie jak typ dołączony do każdego argumentu — wskazuje kompilatorowi, jakiego rodzaju danych ma się spodziewać, gdy funkcja zwróci wartość.



Konstruktory są przypadkiem specjalnym. Może Cię kusić użycie nazwy klasy jako zwracanego typu dla Twojego konstruktora, jednak ActionScript nakazuje pominięcie typu zwracanego dla wszystkich konstruktorów — np. `public function MyClass() {...}`.

Więcej informacji na temat konstruktorów znajdziesz w rozdziale 3.



W AS3 ściśle typowanie zmiennych i funkcji nie jest wprawdzie wymagane, ale wysoce zalecane. Zauważysz, że Twój kod będzie wykonywał się szybciej przy zastosowaniu sprawdzania typów w czasie wykonywania — nie wspominając już o tym, że będzie prostszy w czytaniu, debugowaniu, użyciu i ponownym zastosowaniu.

Jeśli nie jesteś pewien, jakiego typu dane będziesz zwracać, możesz użyć typu danych reprezentowanego przez wieloznacznik (*), który odpowiada danym bez przypisanego typu. Umożliwi to zwracanie dowolnego typu danych.

W przedstawionym poniżej przykładzie typ utworzonych danych jest przekazywany do wywołania funkcji. Jak widzisz, funkcja ta posiada wiele zwracanych typów, ale tylko pierwszy pasujący zostanie wykonany. Reszta nigdy nie zostanie uaktywniona.

```
function createObject(type:String):* {
    switch (type) {
        case "Boolean": return new Boolean();
        case "Number": return new Number();
        case "String": return new String();
        case "Array": return new Array();
        default: trace("Nieznany typ"); return null;
    }
}
var myObject:* = createObject("Array");
myObject.push(1, 2, 3, 4, 5);
trace(myObject); // wyświetli: 1, 2, 3, 4, 5
```

Typ reprezentowany przez wieloznacznik może być w pewnych okolicznościach bardzo pożyteczny, jednak nie powinien go nadużywać. Większość Twoich metod powinna posiadać określony typ danych.

Zwracanie void

Jeśli nie potrzebujesz, żeby Twoja funkcja coś zwracała, użyj typu zwracanego void. Dla funkcji zwracających void nie jest wymagana instrukcja return.

```
function doNothing():void {  
    // Nie będę robić niczego.  
}
```

Nawet jeśli ten przykład „nie robi niczego”, funkcje niezwracające wartości są bardzo powszechnie stosowane. To, że nie zwracają one wartości, wcale nie oznacza, że nie wykonują pożytecznego kodu. W rzeczywistości duża część funkcji w Twoim kodzie prawdopodobnie nie będzie posiadać instrukcji zwracających. Czasami funkcje niezwracające wartości nazywa się **procedurami** — małymi, nadającymi się do wielokrotnego stosowania blokami kodu, które mają na celu zamknięcie w sobie często wykonywanych zadań.



Od ActionScript 3.0 typ danych Void jest pisany małą literą, co służy większej jednolitości z innymi językami programowania. Jeśli przyzwyczaiłeś się do pisania kodu w AS2, uważaj, aby nie popełnić błędu, i nie miej złego samopoczucia, jeśli początkowo kilka takich pomyłek Ci się przytrafi.

Definiowanie funkcji za pomocą wyrażeń funkcyjnych

Do tej pory przyglądaliśmy się definiowaniu funkcji za pomocą instrukcji function. Istnieje jednak alternatywny sposób definiowania funkcji. ActionScript obsługuje również wyrażenia funkcyjne. Posługując się tą techniką, najpierw definiujesz funkcję jako zmienną o typie danych Function, a następnie ustawiasz wartość tej zmiennej na wyrażenie funkcyjne.

```
var doSomething:Function = function(arg:Object):void {  
    // kod funkcji umieszczamy tutaj.  
}
```

Wyrażenia funkcyjne zachowują się trochę odmiennie od instrukcji funkcyjnych. W tabeli 4.1 zamieszczono kilka kluczowych różnic.

Dla większości zastosowań zalecamy używanie w kodzie instrukcji funkcyjnych zamiast wyrażeń funkcyjnych. Spójność i prostota ich stosowania czynią je pierwszorzędym wyborem. Istnieją jednak pewne sytuacje, w których zastosowanie wyrażenia funkcyjnego da Ci elastyczność i pożyteczność, jakiej instrukcja funkcyjna nie potrafi dać. Można do nich zaliczyć:

- ♦ Jednorazowe użycie danej funkcji.
- ♦ Zdefiniowanie funkcji wewnątrz innych funkcji.

Tabela 4.1. *Różnice pomiędzy instrukcjami funkcyjnymi a wyrażeniami funkcyjnymi*

Instrukcja funkcyjna	Wyrażenie funkcyjne
Może być wywoływane z dowolnego miejsca w kodzie, niezależnie od tego, gdzie w kodzie zdefiniowana jest funkcja.	Podobnie jak deklaracje zmiennej dostępne jest tylko po wystąpieniu kodu, w którym funkcja została zdefiniowana.
Łatwiejsza do czytania.	Może być trudna i kłopotliwa w czytaniu.
Może być wywoływana jako metoda za pomocą składni kropkowej.	W trybie standardów nie może być wywoływana jako metoda. Zamiast tego używa się składni kwadratowych nawiasów.
Istnieje w pamięci jako część obiektu, w którym jest zawarta.	Istnieje w pamięci niezależnie. Dobre dla funkcji „do wyrzucenia”, które nie muszą być zachowywane w pamięci na długo.
Nie może zostać nadpisana lub usunięta.	Można mu przypisać nową wartość lub usunąć je za pomocą operatora <code>delete</code> .

- ♦ Dołączenie metody do prototypu klasy.
- ♦ Zmianę funkcjonalności metody w czasie wykonywania.
- ♦ Przekazywanie funkcji jako argumentu dla innej funkcji.

Dostęp do metod superklasy

Zgodnie z tym, czego dowiedziałeś się z rozdziału 3., klasy ActionScript posiadają zdolność rozszerzania funkcjonalności innej klasy. Klasa rozszerzająca funkcjonalność nazywana jest podklasą, a klasa rozszerzana nazywana jest superklasą. Koncepcja ta zwana jest **dziedziczeniem**.

Podczas tworzenia klasy masz do czynienia ze specjalnym mechanizmem, który może zostać wykorzystany do uzyskiwania dostępu do metod superklasy. Jest nim słowo kluczowe `super`. Słowo kluczowe `super` pozwala na wywołanie lub rozszerzenie istniejących metod zdefiniowanych w superklasie. Gdy używane jest samodzielnie, odwołuje się do funkcji konstruktora superklasy. Oznacza to, że może zostać użyte do wywołania całego kodu w funkcji konstruktora superklasy lub klasy rodzica. Jest to najbardziej powszechne zastosowanie.

W poniższym przykładzie kod powinien być zapisany w pliku *MySprite.as*:

```
package {
    import flash.display.*;
    public class MySprite extends Sprite {
        public function MySprite() {
            super();
            x = 100;
            y = 100;
        }
    }
}
```

W tym przykładzie `super()` zostało użyte do wywołania całego kodu funkcji konstruktora klasy `Sprite`, po czym następuje ustawienie `x` i `y` dla `MySprite` na wartość 100.

Obiekt `super` może zostać także użyty do uzyskania dostępu do pozostałych metod superklasy. Użyj po prostu składni `super.method()` do wywołania metody superklasy. W poniższym kodzie *przesłaniamy* metodę `addChild()` klasy `Sprite`, aby wyświetlić obiekt potomka, który dodajemy:

```
package {
    import flash.display.*;
    public class MySprite extends Sprite{
        override public function addChild(child:DisplayObject):DisplayObject {
            var addedChild:DisplayObject = super.addChild(child);
            trace("Właśnie dodany potomek " + addedChild.toString());
            return addedChild;
        }
    }
}
```

Poprzez wywołanie `super.addChild(child)` przekazujemy parametr `child` do metody `addChild()` superklasy. W rezultacie wykonany zostaje oryginalny kod `addChild()`, a jego wynik jest zapisywany do zmiennej `addedChild`. Następnie wyświetlana jest wartość tekstowa `addedChild`. Na końcu jest ona zwracana.

Przesłanianie metod

Być może zwróciłeś uwagę w poprzednim przykładzie na to, iż sygnatura funkcji dla metody `addChild()` została poprzedzona słowem kluczowym `override`. Powoduje to, że definiowana funkcja przesłania lub ponownie definiuje metodę superklasy.

Nowa metoda musi posiadać dokładnie taką samą **sygnaturę** (listę parametrów funkcji, zwracany typ itd.), co metoda przesłaniana. Jeśli nie masz pewności co do dokładnej sygnatury metody, zajrzyj do dokumentacji w celu sprawdzenia budowy tej metody.



Słowo kluczowe `override` musi być umieszczone przed wszystkimi przesłanianymi metodami. Pomińcie go spowodowałoby błąd podczas kompilowania. Metody zdefiniowane za pomocą słowa kluczowego `private` (a w pewnych przypadkach `internal`) nie wymagają słowa kluczowego `override`, ponieważ nie są przekazywane w dół do podklas.

Pisanie funkcji rekurencyjnych

Czasami funkcja w celu zwrócenia wartości musi być wywoływana wielokrotnie, a jej dane wejściowe są zależne od wyniku jej poprzednich wywołań. Taka sprytna forma projektowania funkcji znana jest jako **rekurencja**. Funkcje rekurencyjne są funkcjami wywołującymi siebie wewnątrz swojego ciała funkcji w celu utworzenia wyniku bazującego na wielu poziomach rekurencji. Jest to zaawansowana technika i początkującym programistom może pierwotnie wydawać się dziwna, dlatego jeśli chcesz, możesz pominąć to zagadnienie i powrócić do niego, jeśli kiedykolwiek będziesz musiał dowiedzieć się czegoś więcej na ten temat.

Oto kilka zastosowań, w których mógłbyś użyć funkcji rekurencyjnej:

- ♦ Podczas przemieszczania się po zagnieżdżonych danych, jak w przypadku pliku XML czy listy właściwości.
- ♦ Podczas obliczania funkcji matematycznych, które bazują na wielu iteracjach, jak zagadnienie silni czy obliczenia statystyczne.
- ♦ Funkcje wymagające wracania do poprzednich iteracji lub przechodzące przez złożony zestaw możliwych rozwiązań, jak algorytm poruszania się po labiryncie.

Przyjrzyjmy się często wykorzystywanej funkcji rekurencyjnej — silni. W matematyce funkcja silnia, zazwyczaj zapisywana jako dodatnia liczba całkowita, po której występuje znak wykrzyknika ($4!$), jest funkcją obliczającą iloczyn wszystkich dodatnich liczb całkowitych mniejszych bądź równych danej liczbie całkowitej. Innymi słowy, $3! = 3 \cdot 2 \cdot 1 = 6$. Możemy ująć to w postaci funkcji rekurencyjnej:

```
function factorial(i:uint) {
    if (i == 0) {
        return 1;
    } else {
        return (i * factorial(i - 1));
    }
}
trace(factorial(3)); // wyświetli: 6
```

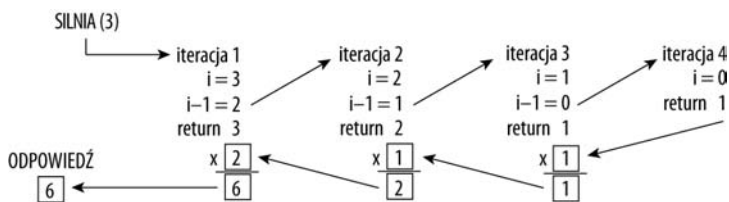
Co się dzieje, gdy wywołujesz tę funkcję? Przekazywana jest początkowa wartość 3. Wartość 3 jest większa od 0, dlatego pomijamy pierwszą instrukcję warunkową i przechodzimy do instrukcji else. Zwraca ona wartość i pomnożoną przez silnię $i-1$ (2). Jednak przed zwróceniem wartości instrukcja silni uruchamiana jest ponownie drugi raz z wykorzystaniem 2 jako parametru. Cały proces rozpoczyna się od nowa. `factorial(2)` zwraca $i * \text{factorial}(i-1)$ (lub 2 razy silnia z 1). Całość powtarzana jest tak długo, aż przetworzone zostaną wszystkie liczby całkowite i zakończy się pętla. Ostatecznie wszystkie zwrócone wartości zostaną wysłane w celu pomnożenia ich przez funkcję.

Leczenie z rekursivitis

Jak mogłeś się przekonać we wcześniejszym przykładzie, praca z funkcjami rekurencyjnymi może szybko się skomplikować, co znacznie utrudni wykrywanie błędów. Nazywam to **objawami rekursivitis** (nie, to nie jest prawdziwy termin medyczny). Jednym z rozwiązań tego problemu, a uważam je za przydatne, jest zapisywanie wartości zmiennych na zwykłej kartce papieru i analizowanie tego, co się wydarzy podczas każdej z iteracji. Dla wcześniej przedstawionego przykładu mógłbym utworzyć graf pokazany na rysunku 4.3.

Rysunek 4.3.

Leczenie z rekursivitis starym, sprawdzonym sposobem



Jak widzisz, dla każdej iteracji śledzę zmiany poprzez ich zapisywanie. Analizowanie kodu w ten sposób jest o wiele prostsze. Z czasem, gdy Twoje zagadnienia będą coraz bardziej złożone, możesz skorzystać z wbudowanego debugera. Narzędzia debugujące opisujemy w rozdziale 20.

Pracując z funkcjami rekurencyjnymi, na pewno przynajmniej raz spotkasz się z problemem przepełnienia stosu. Występuje on wtedy, gdy funkcja nie posiada mechanizmu kończącego jej działanie. Problem podobny do tego to **pętle nieskończone**, które mogą wystąpić podczas pracy z pętlami `for` oraz `while`. Dokładniej opisaliśmy go w rozdziale 2. Flash Player posiada wbudowany mechanizm zatrzymujący każdą pętlę przed wykonaniem zbyt wiele razy. Jeśli Twoja funkcja rekurencyjna powtórzy się zbyt wiele razy, możesz uzyskać taką informację o błędzie:

```
Error: Error #1502: A script has executed for longer than the default timeout period  
↳ of 15 seconds.
```

Funkcje jako obiekty

Warto powtórzyć jeszcze raz: prawie wszystko w ActionScript jest obiektem. Funkcje nie są tutaj żadnym wyjątkiem. Każda funkcja w ActionScript 3.0 jest instancją klasy `Function` z właściwościami i metodami — jak każdy inny obiekt. W tym podrozdziale pokażemy, jak używać tej relacji z korzyścią dla siebie.

Function a function

Słowo *function* pisane małą literą jest ogólnym określeniem, którego używaliśmy do tej pory w tym rozdziale do opisywania wykonywalnego bloku kodu. *Function* pisane dużą literą oznacza klasę `Function`, której instancjami są wszystkie funkcje. Na ogół, tworząc nową instancję klasy, używa się słowa kluczowego `new`, po którym następuje nazwa klasy. Ponieważ funkcje są integralnym i często używanym elementem ActionScript, występują polecenia (jak słowo kluczowe `function`), które używane są zamiast instrukcji `new Function()`.



Technicznie zapis `new Function()` jest poprawny, ale nie robi zbyt wiele poza utworzeniem nowej, niezdefiniowanej funkcji.

Najbardziej prawdopodobne jest, że z nazwą klasy `Function` spotkasz się podczas pracy z metodami wymagającymi funkcji na wejściu lub podczas tworzenia funkcji za pomocą wyrażenia. Np. metoda `addEventListener()` pobiera jako parametr funkcję. Jeśli nie wiesz jeszcze, co to są zdarzenia, nie przejmuj się. Opiszemy je bardzo szczegółowo w części IV tej książki. Teraz przyjrzymy się, jakich argumentów funkcja oczekuje.

```
addEventListener(type:String, listener:Function):void
```



`addEventListener()` akceptuje pięć argumentów. W celu zachowania prostoty dołączono tutaj jedynie dwa wymagane.

Jak widzisz, drugi oczekiwany przez `addEventListener()` parametr jest funkcją nasłuchującą. Jest to funkcja, która zostanie wykonana, gdy wystąpi zdarzenie wskazanego typu (type). Jeżeli chcesz przekazać funkcję jako argument do innej funkcji, użyj nazwy przekazywanej funkcji, ale opuść nawiasy (operator wywołania).

```
function onClick(event:MouseEvent):void {  
    // obsłuż kliknięcie  
}  
addEventListener(MouseEvent.CLICK, onClick);
```



Pamiętaj tutaj o pominięciu nawiasów. Jeśli napiszesz `addEventListener(MouseEvent.CLICK, onClick());`, wtedy funkcja `onClick()` zostanie wykonana i zwróci `void`, co zostanie przekazane jako Twój parametr (`void` oczywiście nie jest funkcją).

Metody i właściwości klasy Function

Klasa `Function` posiada niewiele właściwości i metod. Jako podklasa klasy `Object` dziedziczy wszystkie jej metody i właściwości, jak `toString()`. Oprócz tych dziedziczonych wartości klasa `Function` definiuje swoje własne metody i właściwości. Już przyglądaliśmy się `arguments`, jedynej właściwości definiowanej przez `Function`. Przyjrzyjmy się teraz jej metodom.

Klasa `Function` definiuje dwie metody: `call()` oraz `apply()`, będące alternatywnymi sposobami wywoływania funkcji. Jednak funkcje te nie działają tak samo w ActionScript 3.0, jak działały wcześniej w ActionScript 2.0. W większości przypadków `thisObject` jest kompletnie ignorowany. Zalecamy nie korzystać z tych metod. Jeśli musisz wywołać metody niezdefiniowane w klasie, rozważ użycie klasy `flash.util.Proxy`. Więcej informacji na ten temat znajdziesz w oficjalnej dokumentacji ActionScript 3.0.

Podsumowanie

- ♦ Metody oraz funkcje są powtarzalnymi operacjami wykonującymi bloki kodu w oparciu o przekazane parametry wejściowe.
- ♦ Użycie operatora `return` pozwoli Ci obliczyć i zwrócić wartości w zależności od zmiennych wprowadzonych do Twojej funkcji.
- ♦ Metody — podobnie jak większość innych elementów w ActionScript 3.0 — są obiektami. Należą do klasy `Function`.
- ♦ Funkcje można tworzyć za pomocą instrukcji `function` lub poprzez przypisanie wartości do obiektu typu `Function` za pomocą wyrażeń funkcyjnych.
- ♦ Parametr `...` (reszta) może zostać użyty do uzyskania dostępu do wszystkich argumentów funkcji, jeżeli liczba argumentów jest nieznana lub zmienna.
- ♦ Słowa kluczowe `super` oraz `override` mogą zostać użyte przez podklasę w celu dodania funkcjonalności do metody superklasy.
- ♦ Funkcje rekurencyjne umożliwiają powtarzanie akcji i łączenie wyników wygenerowanych przez jej poprzednie wywołania.