helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

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

Macromedia Flash 8 ActionScript. Oficjalny podręcznik

Autorzy: Jobe Makar, Danny Patterson
Tłumaczenie: Jakub Thiele-Wieczorek na podstawie tłumaczenia Piotra Cieślaka i Marcina Samodulskiego
ISBN: 83-246-0312-3
Tytuł oryginału: Macromedia Flash 8 ActionScript: Training from the Source
Format: B5, stron: 488

Poznaj ogromne możliwości języka ActionScript 2.0

ActionScript to obiektowy język programowania zaimplementowany w jednym z najpopularniejszych narzędzi do tworzenia interaktywnych witryn WWW, czyli w programie Macromedia Flash. Korzystanie z ActionScriptu pozwala wydobyć z Flasha możliwości niedostępne z poziomu jego narzędzi graficznych i animacyjnych. ActionScript daje projektantowi niemal nieograniczoną swobodę twórczą. Za pomocą odpowiednio skonstruowanych skryptów można tworzyć zarówno proste przyciski i animacje, jak i złożone mechanizmy pobierania danych z zewnętrznych źródeł, dynamicznego generowania grafiki i dokumentów oraz sterowania animacją w zależności od działań użytkownika.

Książka „Macromedia Flash 8 ActionScript. Oficjalny podręcznik", przygotowana we współpracy z producentem Flasha – firmą Macromedia, umożliwi Ci odkrycie niesamowitych możliwości języka ActionScript. Wykonując przedstawione w niej ćwiczenia i projekty, poznasz składnię tego języka i zasady programowania obiektowego. Dowiesz się, jak za pomocą skryptów kontrolować odtwarzanie animacji i zachowanie obiektów na scenie oraz nauczysz się korzystać w animacjach z elementów pobieranych z plików i baz danych. Przeczytasz o przetwarzaniu plików XML, weryfikowaniu danych wprowadzanych przez użytkownika oraz odtwarzaniu plików multimedialnych za pomocą Flasha.

- Elementy panelu Actions
- Składnia języka ActionScript
- Funkcje i instrukcje warunkowe
- Klasy obiektów
- Obsługa zdarzeń
- Korzystanie z plików graficznych i cyfrowego wideo
- Tworzenie komponentów
- Odczyt danych z plików zewnętrznych
- Drukowanie z poziomu prezentacji
- Optymalizacja plików SWF

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

Spis treści

O autorach **9**

Wprowadzenie **11**

Lekcja 1. Wprowadzenie do języka ActionScript 2.0 ..... **17**

Czym jest język ActionScript? 18
Korzystanie z panelu Actions 27
Tworzenie zmiennych 38
Sposoby umieszczania kodu w aplikacji 44
Obiekty, klasy i zasięg 47

Lekcja 2. Funkcje **51**

Tworzenie funkcji 52
Funkcje z parametrami 57
Zmienne lokalne oraz funkcje, które zwracają określoną wartość 65

Lekcja 3. Korzystanie z instrukcji warunkowych **73**

Kontrolowanie przebiegu skryptów 74
Reakcja na różne warunki 80
Reakcja na działania użytkownika 82
Wykrywanie krawędzi sceny 83

Lekcja 4. Tablice i pętle **87**

Dlaczego pętle są użyteczne? 88
Typy pętli 89
Wyjątki w pętlach 92
Aplikacja wyszukująca 93
Pisanie i rozumienie warunków pętli 95
Pętle zagnieżdżone 98

Lekcja 5. Posługiwanie się klasami obiektów **103**

Czym są obiekty i dlaczego są przydatne? 104
Zastosowanie klasy Color 119
Klasy String i Selection 123

Lekcja 6. Tworzenie własnych klas obiektów ... 131
Klasy, klasy wierzchołkowe i kopie obiektów ... 132
Tworzenie klasy ... 134
Ścieżka dostępu do definicji klas ... 137
Pakiety i importowanie klas ... 139
Odczytywanie i zapisywanie właściwości klasy ... 144
Definiowanie składowych klasy ... 147
Dziedziczenie ... 151
Modyfikacja projektu wykorzystującego dziedziczenie ... 161

Lekcja 7. Zdarzenia, obiekty typu listener i odwołania ... 167
Do czego służą uchwyty zdarzeń? ... 168
Typy zdarzeń w programie Flash ... 169
Procedury obsługujące zdarzenia ... 169
Tworzenie projektu wykorzystującego procedury obsługujące zdarzenia ... 172
Obiekty typu listener ... 175
Tworzenie projektu wykorzystującego obiekty typu listener ... 176

Lekcja 8. Dynamiczne tworzenie zasobów ... 181
Dodawanie klipów filmowych ... 182
Dodawanie pustych klipów filmowych ... 186
Rysowanie za pomocą kodu ... 187
Wykorzystywanie klipów filmowych jako przycisków ... 190
Dodawanie pól tekstowych za pomocą kodu ... 192
Formatowanie tekstu ... 194

Lekcja 9. Wykorzystywanie grafiki rastrowej ... 197
Buforowanie powierzchni bitmapowych ... 198
Wykorzystywanie filtrów ... 203
Tryby przenikania ... 211
Wprowadzenie do Bitmap API ... 214

Lekcja 10. Tworzenie komponentów interfejsu użytkownika ... 223
Komponenty: elementarz pisania skryptów ... 225
Konfiguracja właściwości komponentów ... 226
Wyzwalanie skryptów przy użyciu zdarzeń komponentów ... 233
Korzystanie z metod komponentów ... 239
Korzystanie z komponentu FocusManager ... 248
Stylizacja komponentów interfejsu użytkownika z wykorzystaniem języka ActionScript ... 252

Lekcja 11. Zaawansowane programowanie obiektowe ... 257
Usuwanie błędów zakresu za pomocą delegacji ... 258
Enkapsulacja ... 262
Wykorzystywanie kompozycji ... 263
Łączenie kompozycji i dziedziczenia ... 265
Wyzwalanie zdarzeń ... 268

Lekcja 12. Walidacja danych ... 275
Proces walidacji danych ... 276
Korzystanie z procedur walidacyjnych ... 277
Obsługa błędów ... 280
Walidacja ciągów znaków ... 282
Walidacja sekwencji ... 287
Walidacja z wykorzystaniem listy możliwych wartości ... 291
Walidacja liczb ... 294
Przetwarzanie danych po procesie walidacji ... 296

Lekcja 13. Połączenia z zewnętrznymi źródłami danych ... 299
Źródła i formaty danych ... 300
Instrukcje GET i POST ... 303
Korzystanie z klasy LoadVars ... 304
Pliki reguł ... 313
Korzystanie z obiektów udostępnionych ... 315
Korzystanie z komponentu WebServiceConnector ... 326

Lekcja 14. Korzystanie z języka XML w programie Flash ... 335
Podstawy języka XML ... 337
Korzystanie z klasy XML ... 340
Korzystanie z serwerów gniazd ... 349

Lekcja 15. Interfejs zewnętrzny ... 367
Klasa ExternalInterface ... 368
Konfiguracja języka HTML dla potrzeb prostych wywołań klasy ExternalInterface ... 369
Wywoływanie funkcji języka JavaScript w ActionScripcie ... 369
Korzystanie z możliwości języka JavaScript w programie Flash ... 370
Wywoływanie funkcji języka ActionScript w JavaScripcie ... 373
Aplikacja-quiz ... 374

Lekcja 16. Korzystanie z dźwięku i filmów **381**
Sterowanie odtwarzaniem dźwięków za pomocą ActionScriptu 383
Tworzenie kopii klasy Sound 383
Przeciąganie klipu filmowego na określonym obszarze 386
Sterowanie głośnością 390
Sterowanie balansem 397
Dołączanie dźwięków i sterowanie ich odtwarzaniem 402
Wczytywanie i kontrolowanie odtwarzania zewnętrznych plików wideo 407

Lekcja 17. Drukowanie i tworzenie podręcznych menu **417**
Różnice między drukowaniem z poziomu Flasha a przeglądarki 418
Korzystanie z klasy PrintJob 420
Tworzenie własnych menu podręcznych 429

Lekcja 18. SWF o maksymalnych możliwościach **437**
Polecenie fscommand() 438
Korzystanie z programu Zinc 444
Użycie mechanizmu przekazywania parametrów FlashVars do filmu 458

Skorowidz **467**

2 Funkcje

Podczas programowania niekiedy spotykasz się z koniecznością wprowadzenia wielokrotnie powtarzających się części kodu — wówczas możesz uprościć sobie zadanie, kopiując je i wklejając lub przepisując te same linie w edytorze ActionScript. Jest jednak sposób na to, by powtarzające się fragmenty napisać tylko raz, a przy najbliższych okazjach zastępować je pojedynczym poleceniem. Sposób ten polega na tworzeniu *funkcji*, zaś wykonanie funkcji nosi nazwę *wywołania funkcji* lub po prostu *wywołania*. Użycie funkcji oszczędza mnóstwo czasu zarówno podczas pisania, jak i modyfikowania aplikacji. Zmniejszają one długość kodu niezbędnego do jej działania, poprawiając przejrzystość i przyspieszając ewentualne zmiany.

W czasie tej lekcji wykonamy ćwiczenie polegające na zaprojektowaniu prostej aplikacji naśladującej wirtualny odbiornik TV. Za pomocą funkcji zrealizujemy przełączanie programów oraz włączanie i wyłączanie odbiornika Flash-TV

Potraktuj funkcje jako miniprogramy, które spełniają określoną rolę wewnątrz większej aplikacji. Możesz wykorzystać je do wykonania zestawu określonych poleceń, przetworzenia przekazanych im danych i zwrócenia rezultatu lub obu tych działań jednocześnie. Funkcje stanowią bardzo elastyczną i wygodną technikę programowania.

Podczas tej lekcji wykonamy jedno ćwiczenie, w którym nauczysz się, w jaki sposób tworzyć funkcje i z nich korzystać.

Czego się nauczysz?

W czasie tej lekcji:

- Utworzysz funkcję
- Odwołasz się do niej
- Zaprojektujesz funkcję korzystającą z parametrów
- Napiszesz funkcję zwracającą określony rezultat
- Skorzystasz ze zmiennych lokalnych

Czas trwania

Lekcja zajmie Ci około półtorej godziny.

Materiały do lekcji

Pliki multimedialne:

Brak

Pliki startowe:

Lekcja02\Start\television1.fla

Gotowy projekt:

Lekcja02\Completed\television4.fla

Tworzenie funkcji

Zanim zaczniesz korzystać z funkcji, musisz ją utworzyć lub zdefiniować. Można tego dokonać, posługując się jedną z dwóch dostępnych konstrukcji.

Konstrukcja 1.

Poniższy kod przedstawia pierwszą:

```
function myFunction (parameter1:DataType,parameter2:DataType,etc.) {
  // W tym miejscu znajdują się akcje;
}
```

Skrypt ten odzwierciedla najpopularniejszy sposób tworzenia funkcji, jest to również metoda, którą będziemy posługiwać się najczęściej w trakcie bieżącej lekcji. Jak widać, deklaracja funkcji rozpoczyna się od słowa kluczowego `function`, za którym następuje nazwa funkcji (dowolna wybrana

przez Ciebie, lecz zgodna z ogólnymi zasadami nazewnictwa obowiązującymi we Flashu). Część definicji znajdująca się za nazwą została ujęta w nawiasy. Możesz pozostawić ją pustą lub wypełnić odpowiednimi wyrażeniami (będą to parametry), które funkcja będzie wykorzystywać do swojego działania. Pozostawiając puste nawiasy, tworzysz tzw. funkcję „ogólną" — czyli taką, która jest wykonywana zawsze w ten sam sposób, niezależnie od tego, kiedy zostanie wywołana (czyli uruchomiona). Jeśli Twoja funkcja przyjmuje parametry, będzie wykonana w inny sposób za każdym jej wywołaniem; sposób ten będzie zależał od wartości i rodzaju parametrów, które jej wyślemy. Takie wysyłanie informacji do funkcji nosi nazwę *przekazywania argumentów* lub *przekazywania parametrów*. Możesz wykorzystać dowolną liczbę parametrów. W dalszej części lekcji napiszemy, w jaki sposób korzystać z danych przekazywanych przez parametry.

Za opcjonalnym elementem deklaracji funkcji, jakim są parametry, następuje otwarcie nawiasu klamrowego oraz powrót karetki (przejście do nowego wiersza). Jest to miejsce przeznaczone na wszelkie polecenia i działania, które ma podjąć funkcja podczas wywołania. Polecenia te mogą wykorzystać dowolny z parametrów przekazany do funkcji (o czym wkrótce się przekonasz).

Konstrukcja 2.

Poniższy fragment skryptu stanowi odzwierciedlenie drugiej konstrukcji przeznaczonej do tworzenia funkcji.

```
myFunction = function (parameter1:DataType,parameter2:DataType,etc.)
  {/* W tym miejscu znajdują się akcje */
};
```

Konstrukcji tej można użyć do dynamicznego tworzenia funkcji lub podczas definiowania własnej *metody* (zagadnienie to poruszymy podczas lekcji 6., „Tworzenie własnych klas obiektów"). Jedyna różnica, jaka dzieli obydwie przedstawione składnie, polega na sposobie nadania funkcji nazwy: w tym wypadku nazwa ta umieszczona jest na początku konstrukcji, zaś między nią a definicją działania i operatorami funkcji znajduje się znak przypisania (=).

Po zapoznaniu się z podstawami tworzenia czy też definiowania funkcji przyjrzyjmy się sposobowi ich wykorzystania lub, mówiąc inaczej, wywoływania.

Jeśli definicja funkcji nie zawiera żadnych parametrów, może ona zostać wywołana przez zastosowanie następującej konstrukcji:

```
myFunction();
```

Wywołując określoną funkcję, tym samym przekazujesz do Flasha instrukcję wykonania poleceń w niej zawartych. Jeśli funkcja `myFunction()` składałaby się z 20 poleceń, wszystkie one zostałyby wykonane dzięki tej pojedynczej linii skryptu.

Jeżeli definicja funkcji obejmuje również przyjmowanie parametrów działania, do jej wywołania możesz wykorzystać następującą konstrukcję:

```
myFunction(parameter1, parameter2);
```

Jeśli parametr `parameter1` miałby wartość w postaci słowa `cat`, zaś parametrem `parameter2` byłaby liczba `36`, to te dwie wartości zostałyby przekazane do funkcji i mogłyby zostać wykorzystane wewnątrz

niej do przeprowadzenia zdefiniowanych wcześniej operacji i działań. W innym miejscu skryptu ta sama funkcja mogłaby zostać wywołana z zupełnie innymi parametrami. Sprawiłoby to, że ta sama funkcja zadziałałaby nieco inaczej, gdyż polecenia zawarte w ciele funkcji przy zmienionych parametrach dałyby odmienny wynik.

W kolejnych przykładach poczyniliśmy założenie, że zarówno funkcja, jak i odwołania do niej znajdują się na tej samej listwie czasowej. Wiesz już, że każda listwa czasowa może zawierać własne obiekty i zmienne; dokładnie to samo tyczy się funkcji, które zostały w niej zdefiniowane. Kiedy wywołujesz funkcję znajdującą się na określonej listwie czasowej, musisz wskazać jej położenie, poprzedzając nazwę funkcji ścieżką dostępu do tej listwy. Może to wyglądać np. tak:

```
_root.clip1.clip2.myFunction();
```

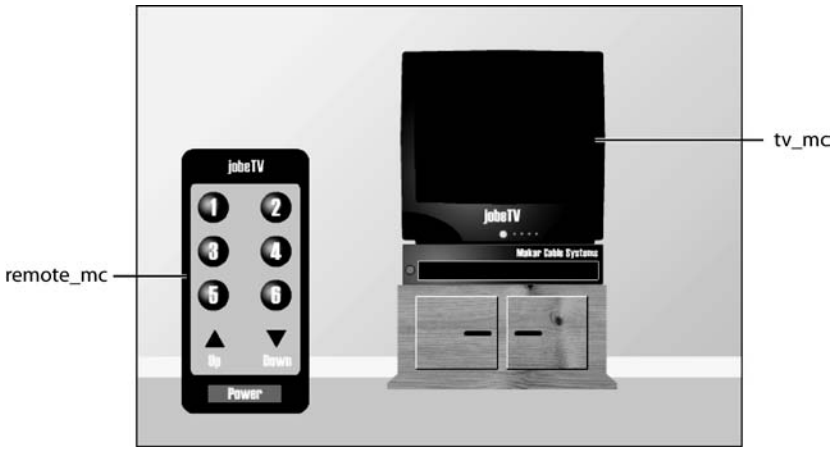
W ćwiczeniu tym zaprojektujemy przycisk, spełniający rolę włącznika na pilocie zdalnego sterowania telewizora. Za pomocą tego przycisku nasz odbiornik Flash TV będzie można włączać lub wyłączać. Podczas pisania skryptów skorzystamy z możliwości oferowanych przez funkcje.

Uwaga

Funkcja może być wywołana dynamicznie w oparciu o podaną wartość, np. _root[aVariableName]();. W takiej konstrukcji, jeśli aVariableName miałaby wartość "sayHello", wynikowe odwołanie do funkcji przybrałoby postać _root.sayHello();.

1. Otwórz plik *television1.fla* znajdujący się w folderze *Lekcja02/Starts*.

   Struktura filmu została już przygotowana. Cały kod ActionScript, jaki napiszesz w tym ćwiczeniu, zostanie umieszczony w warstwie *Actions* projektu. W warstwie o nazwie *TV* znajduje się kopia klipu filmowego o nazwie *tv_mc*. Listwa czasowa tego klipu składa się z trzech warstw: w widocznej na samym dole warstwie *Television* umieszczono rysunek telewizora, zaś warstwa znajdująca się ponad nią (*Screen*) zawiera kopię klipu o nazwie *screen_mc* (który zamaskowany jest przez warstwę umieszczoną na samej górze). Kopia klipu *screen_mc* składa się z dwóch warstw i dwóch klatek, a zawiera grafikę naśladującą „programy" przełączane za pomocą pilota TV.



   Warstwa *Remote* składająca się na główną listwę czasową zawiera kopię klipu filmowego o nazwie *remote_mc*. Wewnątrz tego klipu znajdziesz warstwę, w której umieszczono większość

elementów graficznych przedstawiających pilota, włączając w to kopię klipu o nazwie *light_mc* oraz kolejną warstwę, na której widnieją przyciski zdalnego sterowania. Ponumerowane przyciski noszą nazwy od *channel1_btn* do *channel6_btn*. Pod przyciskami zaopatrzonymi w cyfry widoczne są dodatkowe dwa, oznaczone napisami *Up* oraz *Down*. Nazwy kopii tych przycisków to, odpowiednio, *up_btn* oraz *down_btn*. Znajdujący się na samym dole przycisk z napisem *Power* nosi nazwę *power_btn*.

2. Zaznacz 1. klatkę warstwy *Actions* na głównej listwie czasowej. Korzystając z panelu *Actions*, wprowadź następujący skrypt:

```
var tvPower:Boolean = false;
function togglePower() {
  var newChannel:Number;
  if (tvPower) {
    newChannel = 0;
    tvPower = false;
  } else {
    newChannel = 1;
    tvPower = true;
  }
  tv_mc.screen_mc.gotoAndStop(newChannel+1);
  remote_mc.light_mc.play();
}
```

W pierwszej linii tego skryptu deklarowana jest zmienna o nazwie `tvPower`, która posłuży do śledzenia bieżącego stanu odbiornika TV. Jeśli będzie ona miała wartość `true`, to znaczy, że telewizor jest włączony, wartość `false` oznacza wyłączenie odbiornika. Ponieważ po uruchomieniu filmu telewizor jest wyłączony, zmiennej `tvPower` nadawana jest wartość `false`.

Kolejnych 11 linii kodu zawiera definicję funkcji `togglePower()`. Funkcja ta po wywołaniu spowoduje włączenie lub wyłączenie telewizora. Do działania funkcji nie są wymagane żadne parametry. Ponieważ skrypt ten znajduje się w 1. klatce filmu, funkcja `togglePower()` oraz zmienna `tvPower` z nadaną wartością `false` są deklarowane natychmiast po wczytaniu tej klatki, czyli rozpoczęciu odtwarzania filmu.

Uwaga

Ponieważ funkcje muszą być zdefiniowane, zanim nastąpi ich pierwsze wywołanie, standardową praktyką jest definiowanie ich w pierwszych klatkach filmu, dzięki czemu mogą one zostać wywołane w dowolnym, późniejszym momencie.

Pierwsza część funkcji zawiera wyrażenie warunkowe `if`, analizujące bieżącą wartość zmiennej `tvPower`. Jeśli zmienna ta przechowuje wartość `true` (TV jest włączony), to po wywołaniu funkcji zostaje ona zmieniona na `false` (wyłączenie TV), zaś zmiennej `newChannel` przechowującej numer bieżącego kanału zostaje nadana wartość `0`. Jeżeli zmienna `tvPower` ma wartość `false`, to (kod za wyrażeniem `else`) zostaje ona zmieniona na `true`, zaś zmiennej `newChannel` przypisywana jest wartość `1`. Stosując wyrażenie `if` w taki właśnie sposób, osiągamy efekt zmiany wartości zmiennej `tvPower` na przeciwną, za każdym razem gdy wywołana jest funkcja `togglePower()`. Jednocześnie następuje również zmiana wartości zmiennej `newChannel`. Po zakończeniu działania skryptu zmienna ta przyjmuje wartość `0` lub `1`.

Uwaga

W tej lekcji wykorzystamy kilkakrotnie instrukcje if. Jednak ich dokładny opis znajdziesz dopiero w lekcji 4., "Tablice i pętle". W tej chwili wystarczy, że będziesz wiedział, iż są one używane do sprawdzania, czy został spełniony określony warunek. Jeśli tak, wykonywany jest dodatkowy kod.

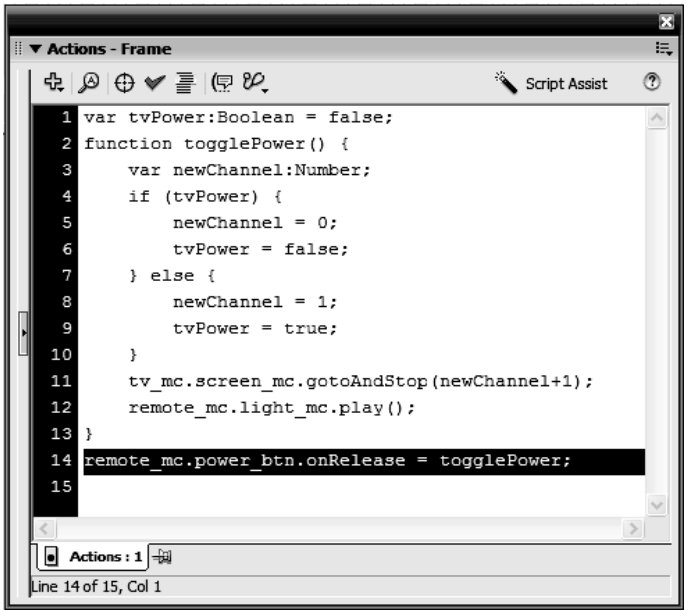
Po zakończeniu opisanych powyżej działań funkcja powoduje odtworzenie odpowiedniej klatki kopii klipu filmowego *screen_mc*, który znajduje się wewnątrz klipu *tv_mc*. Numer tej klatki uzyskany jest przez zwiększenie wartości zmiennej `newChannel` o `1`. Zabieg ten jest konieczny, ponieważ zapobiega odtworzeniu klatki o numerze 0 (mogłoby się tak zdarzyć, ponieważ zmienna `newChannel` przyjmuje niekiedy wartość `0`, a klatki o takim numerze nie zawiera żadna listwa czasowa Flasha). Podsumowując działanie omawianej części funkcji, stwierdzamy, że powoduje ona odtworzenie klatki 1. (pusty ekran TV) lub 2. (program 1.) kopii klipu filmowego *screen_mc*.

Funkcja kończy działanie na przekazaniu polecenia odtwarzania klipowi przedstawiającemu nadajnik zdalnego sterowania. Powoduje to błyśnięcie „diody" umieszczonej na pilocie, co stanowi potwierdzenie wciśnięcia przycisku.

Dysponujemy zatem funkcją zdefiniowaną w 1. klatce głównej listwy czasowej filmu. Choć funkcja ta składa się z kilku poleceń, żadne z nich nie jest wykonywane aż do momentu jej wywołania.

3. Korzystając z panelu *Actions*, dodaj kolejny fragment skryptu poniżej już wprowadzonego (po definicji funkcji)

```
remote_mc.power.btn.onRelease = togglePower;
```



Przycisk *Power* umieszczony na pilocie zawarty jest w kopii klipu filmowego o nazwie *power_btn* (czyli jednocześnie znajduje się również w klipie *remote_mc* z pilotem). Linia kodu, którą przed chwilą wpisałeś, przypisuje uchwyt zdarzenia `onRelease` do przycisku *Power*. Po zajściu tego zdarzenia wywoływana jest nasza funkcja `togglePower`. Dzieje się tak za każdym wciśnięciem i zwolnieniem tego przycisku, co powoduje wykonanie poleceń zawartych w funkcji.

4. Wydaj polecenie *Control/Test Movie*. Kliknij kilkukrotnie przycisk *Power*, aby przekonać się, że włączanie i wyłączanie wirtualnego telewizora działa bez zarzutu.

   Za każdym razem gdy wciskasz przycisk *Power*, wywoływana jest funkcja `togglePower()`, a wraz z nią wykonywane są wszystkie polecenia w niej zawarte. Zgodnie z tym, co mówiliśmy przed chwilą, powodują one naprzemienne włączanie i wyłączanie TV.

5. Zakończ testowanie filmu i zapisz projekt w pliku o nazwie *television2.fla*.

   Właśnie utworzyłeś i zastosowałeś swoją pierwszą funkcję! W następnej części lekcji rozbudujemy nieco projekt, aby zaprezentować działanie bardziej rozbudowanej i elastycznej funkcji.

Funkcje z parametrami

W naszym poprzednim ćwiczeniu nauczyłeś się, w jaki sposób zdefiniować funkcję i wywołać ją. W ćwiczeniu, które za chwilę rozpoczniemy, funkcję tę wzbogacimy w możliwość wykorzystania parametrów, przedstawimy też opis ich zastosowania. Oto konstrukcja pozwalająca na zdefiniowanie funkcji, do której można przekazać parametry:

```
function convertToMoonWeight (myWeight:Number) {
  var weightOnMoon:Number = myWeight/6.04;
}
```

Jedyny parametr funkcji zdefiniowanej w tym skrypcie nosi nazwę `myWeight`. Wartość tego parametru wykorzystana jest także *wewnątrz* definicji funkcji (pod koniec drugiej linii kodu), zupełnie jakby była to zdefiniowana wcześniej zmienna. Zauważ, że definicja funkcji z parametrami powinna zawierać deklaracje typów tych parametrów. W tym przypadku parametr o nazwie `myWeight` jest wartością numeryczną.

Oto przykład wywołania zdefiniowanej w ten sposób funkcji:

```
convertToMoonWeight(165);
```

Jak widzisz, do wywołania funkcji dodaliśmy wartość numeryczną w postaci liczby `165`. Wartość ta wysyłana jest do wywoływanej funkcji, dzięki czemu może ona zostać wykorzystana do obliczeń zdefiniowanych wewnątrz tej funkcji. W zaprezentowanym tutaj przykładzie wartość `165` umieszczona w *wywołaniu funkcji* przypisywana jest do parametru `myWeight` w *definicji funkcji*. Rezultat wygląda następująco:

```
function convertToMoonWeight (165) {
  var weightOnMoon:Number = 165/6.04;
}
```

W taki oto sposób w naszym przykładzie wysłanie wartości `165` do funkcji `convertToMoonWeight()` spowoduje obliczenie wyniku zmiennej `weightOnMoon`, której wartość wyniosłaby 165/6.04, czyli `27.32`.

Parametr `myWeight` zastępowany jest wartością przekazywaną w odwołaniu do funkcji. Największa zaleta tego mechanizmu polega na możliwości wysłania za każdym razem innej wartości podczas wywołania tej samej funkcji. Dzięki temu zmienna `weightOnMoon` obliczona na podstawie przekazanej liczby również będzie miała inną wartość. Przyjrzyjmy się różnym odwołaniom do funkcji `convertToMoonWeight()`:

```
convertToMoonWeight(190);
convertToMoonWeight(32);
convertToMoonWeight(230);
```

Każde z tych odwołań dotyczy tej samej funkcji, jednak ze względu na to, że przekazywany do niej parametr ma za każdym razem inną wartość, także i efekt jej działania będzie różny (mówiąc innymi słowy, inny będzie przechowywany w zmiennej `weightOnMoon` rezultat obliczeń zdefiniowanych wewnątrz funkcji).

Uwaga

Parametry określone w definicji funkcji istnieją jedynie w jej obrębie. W naszym przykładzie parametrem takim jest wartość myWeight, która nie jest dostępna poza samą funkcją.

Wysyłając do funkcji określone wartości parametrów, możesz skonstruować wywołanie zawierające nazwę zmiennej, np.:

```
convertToMoonWeight(myVariable);
```

W takim wypadku do funkcji zostanie wysłana wartość zmiennej `myVariable`.

Definicje funkcji mogą też obejmować kilka parametrów, np.:

```
function openWindow(url:String, window:String) {
getURL(url, window);
}
```

Podana definicja funkcji zawiera dwa parametry, `url` oraz `window`, oddzielone przecinkiem. Funkcja ta składa się z jednego polecenia, `getURL()`, które wykorzystuje oba przekazane do funkcji parametry. Odwołanie do tej funkcji mogłoby wyglądać np. tak:

```
openWindow("http://www.yahoo.com", "_blank");
```

Odwołanie to *wysyła* do funkcji dwie wartości, oddzielone przecinkiem; jest to adres URL oraz sposób otwarcia strony definiowany przez atrybut *target*. Wartości te wykorzystane zostaną przez funkcję do zrealizowania określonego zadania, w tym przypadku będzie to otwarcie witryny *yahoo.com* w nowym oknie przeglądarki.

Deklarując wiele parametrów w definicji funkcji, należy zapamiętać kolejność, w jakiej zostały podane. Wartości umieszczone w wywołaniu muszą być bowiem ułożone w tej samej kolejności, w jakiej zdefiniowano odpowiadające im parametry.

Przykładowo poniższe odwołanie do zaprezentowanej wcześniej funkcji `openWindow()` nie zadziała, ponieważ definicja funkcji zakłada podanie najpierw adresu URL, a dopiero potem sposobu otwarcia strony:

```
openWindow("_blank", "http://www.yahoo.com");
```

Uwaga

Po wywołaniu funkcji tworzona jest pomocnicza, tymczasowa tablica o nazwie arguments. Tablica ta zawiera wszystkie parametry przekazane do funkcji nawet wtedy, jeśli definicja funkcji nie zawiera żadnych parametrów. Oto przykład ilustrujący sposób, w jaki można uzyskać dostęp do tablicy arguments:

```
function traceNames() {
  trace("This function was passed " + arguments.length + "arguments");
  trace ("The value of the first argument is: " + arguments[0]);
  trace ("The value of the second argument is: " + arguments[1]);
}
traceNames("Kelly", "Chance");
```

Po wykonaniu przykładowego skryptu w oknie output Flasha wyświetlone zostaną następujące informacje:

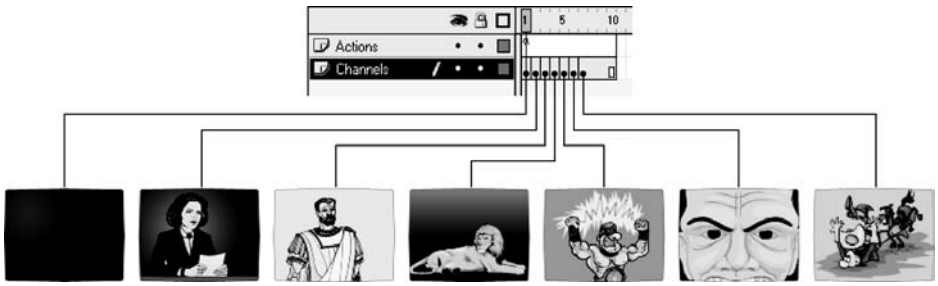
```
This function was passed two arguments
The value of the first argument is: Kelly
The value of the second argument is: Chance
```

Dostęp do tablicy `arguments` pozwala na tworzenie funkcji, które mogą przystosowywać się do zmiennej ilości parametrów. Więcej informacji na temat tablic znajdziesz w lekcji 4., „Tablice i pętle".

W ćwiczeniu, które za chwilę rozpoczniemy, rozbudujemy nieco nasz poprzedni projekt, dodając funkcje do przycisków numerycznych oraz przycisków zmiany kanałów *Up* oraz *Down* umieszczonych na pilocie telewizora. Korzystając z tych przycisków, będziemy mogli zmienić wyświetlany kanał telewizyjny. Działanie przycisków numerycznych oprzemy się na funkcji, której przekazywać będziemy parametr w postaci numeru żądanego kanału. Nieznacznym zmianom poddamy również funkcję `togglePower()`.

1. Otwórz plik *television2.fla*.

   W ćwiczeniu tym wykorzystamy plik, który zapisałeś na końcu pracy z poprzednim projektem. Zanim jednak przystąpimy do dalszej pracy, powinieneś zapoznać się ze strukturą kopii klipu *screen_mc*, znajdującą się wewnątrz klipu *tv_mc*. W klatkach od 1. do 7. listwy czasowej klipu *screen_mc* znajdują się elementy graficzne. Odzwierciedlają one wygląd ekranu, gdy telewizor jest wyłączony (klatka 1.) oraz sześć kolejnych kanałów telewizyjnych, które można będzie włączyć za pomocą pilota.



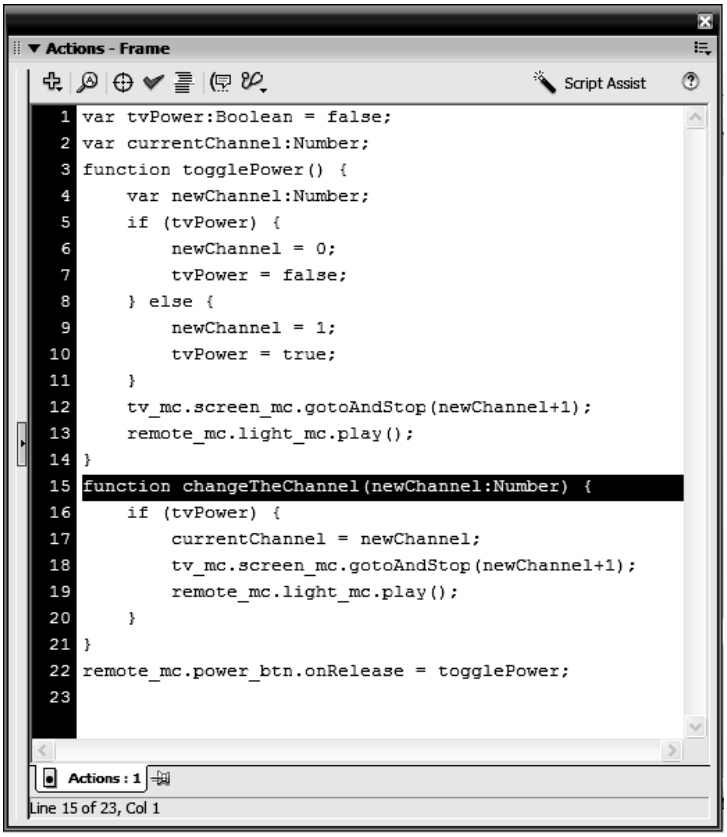
2. Wybierz 1. klatkę na warstwie *Actions* głównej listwy czasowej i otwórz panel *Actions*. Poniżej linii `var tvPower:Boolean = false;` dodaj następujący fragment skryptu:

```
var currentChannel:Number;
```

W ćwiczeniu tym zajmiemy się przygotowaniem funkcji zmieniającej kanał wyświetlany w TV. Zaprogramujemy też zmianę kanału za pomocą przycisków *Up* — „następny" oraz *Down* — „poprzedni". W tym celu musimy dysponować informacją o bieżącym kanale wyświetlanym w TV. Przedstawiona linia skryptu deklaruje nową zmienną o nazwie `currentChannel`, która będzie służyła do przechowywania numeru bieżącego, wyświetlanego kanału.

3. W tej samej klatce filmu, tuż pod znajdującą się tam definicją funkcji, dodaj kolejny fragment skryptu:

```
function changeTheChannel(newChannel:Number) {
  if (tvPower) {
    currentChannel = newChannel;
    tv_mc.screen_mc.gotoAndStop(newChannel+1);
    remote_mc.light_mc.play();
  }
}
```



Zdefiniowałeś właśnie funkcję z parametrem. Funkcja ta powoduje zmianę bieżącego kanału TV na podstawie wartości przekazanego do niej parametru (`newChannel`). Wszystkie polecenia realizowane przez funkcję znajdują się wewnątrz wyrażenia warunkowego `if`, które pozwala na zmianę kanałów *wyłącznie* wtedy, gdy zmienna `tvPower` ma wartość `true`. Kolejne polecenie przypisuje zmiennej przechowującej bieżący numer kanału wartość parametru, który przekazałeś do funkcji.

Kolejne dwie linie skryptu powinny wyglądać znajomo: omawialiśmy je w poprzednim ćwiczeniu przy okazji definiowania funkcji `togglePower()`. Powodują one odtworzenie odpowiedniej klatki klipu *screen_mc* (a co za tym idzie, zmianę kanału TV) oraz błyśnięcie diody kontrolnej na pilocie. Rozważmy następujący przykład, który pomoże w zrozumieniu działania omawianej funkcji. Załóżmy, że wykonane zostało następujące odwołanie:
`changeTheChannel(4);`

Na początku funkcja sprawdzi, czy zmiennej `tvPower` przypisano wartość `true` (TV jest włączony). Jeśli tak, to zmiennej `currentChannel` nadana zostanie wartość `4` (taka sama jak wartość parametru przekazanego do funkcji). Następnie odtworzona zostanie klatka klipu *screen_mc* o numerze równym tej wartości, zwiększonej o 1 (czyli `4 + 1`). W omawianym przypadku będzie to zatem klatka 5.

Twoja nowa funkcja jest gotowa do wykorzystania. Dodajmy jeszcze uchwyty zdarzeń `onRelease` do każdego z przycisków numerycznych pilota, pozwolą one wywoływać funkcję `changeTheChannel()`.

4. Dodaj następujący skrypt poniżej już wprowadzonego w 1. klatce głównej listwy czasowej:

```
remote_mc.channel1_btn.onRelease = function() {
  changeTheChannel(1);
};
remote_mc.channel2_btn.onRelease = function() {
  changeTheChannel(2);
};
remote_mc.channel3_btn.onRelease = function() {
  changeTheChannel(3);
};
remote_mc.channel4_btn.onRelease = function() {
  changeTheChannel(4);
};
remote_mc.channel5_btn.onRelease = function() {
  changeTheChannel(5);
};
remote_mc.channel6_btn.onRelease = function() {
  changeTheChannel(6);
};
```

Do każdego z przycisków numerycznych na pilocie dodany został w ten sposób uchwyt zdarzenia (przypominamy, że kopia klipu *remote_mc* zawiera sześć przycisków o nazwach od *channel1_btn* do *channel6_btn* — nazwy te stanowią podstawę dla zastosowanej konstrukcji). Gdy jeden z tych przycisków zostanie wciśnięty, nastąpi odwołanie do funkcji `changeTheChannel()` i przekazanie jej numeru kanału. W ten sposób możemy wykorzystać tę samą funkcję w odniesieniu do kilku różnych przycisków i uzyskać rezultat zależny od przekazanego jej parametru.

Uwaga

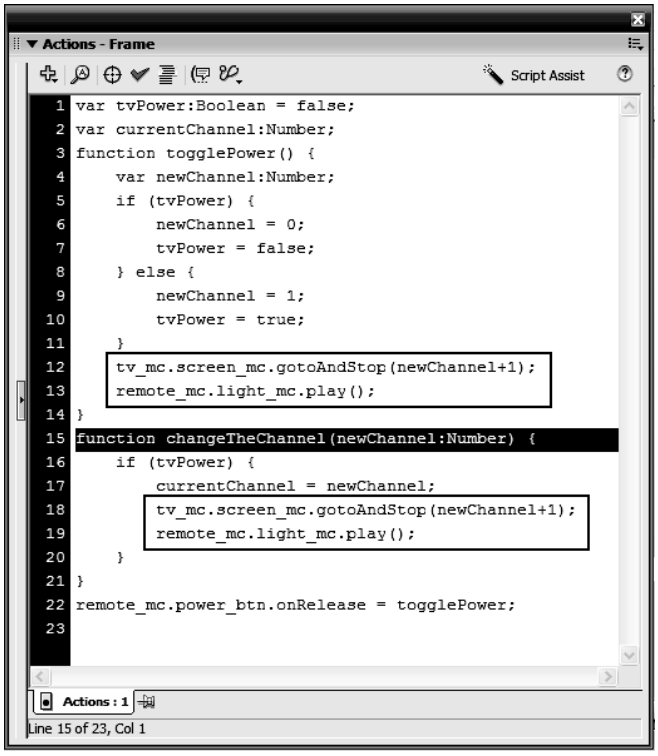
Funkcje zdefiniowane za pomocą operatora przypisania (=) traktowane są jak polecenia. Wszystkie polecenia powinny być zakończone średnikami. Dlatego też każda przedstawiona tutaj definicja funkcji zakończona jest średnikiem, choć standardowa konstrukcja definicji nie wymaga takiego zakończenia.

5. Wydaj polecenie *Control/Test Movie*. Wciśnij przycisk *Power* na pilocie, aby włączyć telewizor, a następnie skorzystaj z przycisków oznaczonych cyframi do przełączania programów TV.

Jeżeli próbowałeś wcisnąć dowolny przycisk z numerem programu, zanim włączyłeś odbiornik, funkcja `changeTheChannel()` nie wykonała żądanej zmiany. Dopiero po włączeniu telewizora i wciśnięciu jednego z przycisków kanałów liczba, którą oznaczony jest przycisk, przekazywana jest w charakterze parametru do funkcji `changeTheChannel()` i odtwarzanie kopii klipu *screen_mc* zostaje przeniesione do odpowiedniej klatki (kanału).

6. Zakończ testowanie pliku i powróć do środowiska edycji projektu. Zaznacz 1. klatkę warstwy *Actions* głównej listwy czasowej.

W klatce tej znajdują się teraz dwie definicje funkcji, jedna powodująca włączenie i wyłączenie TV, druga zaś pozwalająca na zmianę programów za pomocą przycisków numerycznych na pilocie telewizora. Przyglądając się konstrukcji obu funkcji, możemy jednak zauważyć pewne nadmiarowe, powtarzające się elementy. Obydwie zawierają polecenia powodujące błyskanie lampki kontrolnej pilota, obie też powodują odtworzenie wskazanej klatki klipu *screen_mc*. Takie powtarzające się fragmenty najlepiej eliminować zawsze, gdy to tylko możliwe, zajmiemy się więc teraz poprawieniem tego problemu.



7. Korzystając z panelu *Actions*, zmień funkcję `togglePower()` w następujący sposób:

```
function togglePower() {
  if (tvPower) {
    changeTheChannel(0);
    tvPower = false;
  } else {
    tvPower = true;
    changeTheChannel(1);
  }
}
```

Funkcja `togglePower()` wykorzystuje teraz odwołanie do funkcji `changeTheChannel()`, aby zmienić bieżący program, gdy telewizor jest włączany lub wyłączany. Przy włączeniu, podczas odwołania do funkcji `changeTheChannel()`, przekazywana jest wartość `1`. Oznacza to, że za każdym razem gdy telewizor będzie włączany, automatycznie wyświetlany będzie program 1. Po wyłączeniu telewizora „wyświetlany" będzie kanał 0 (czyli pusty ekran). Stanowi to prosty przykład ilustrujący możliwość wywołania funkcji z poziomu innej funkcji.

Uwaga

Przyjrzyj się pierwszej części wyrażenia if w punkcie 7. ćwiczenia i zwróć uwagę, że odwołanie do funkcji changeTheChannel() ma miejsce, zanim jeszcze zmiennej tvPower przypisana zostanie wartość false. Kolejność taka spowodowana jest konstrukcją funkcji changeTheChannel(), która wykonana zostanie jedynie wówczas, gdy zmienna tvPower ma wartość true (a niewątpliwie ma taką wartość, ponieważ jest ona sprawdzana przez wyrażenie if). Jeżeli przypisalibyśmy zmiennej tvPower wartość false, zanim nastąpiłoby wywołanie changeTheChannel(0), funkcja ta nie wykonywałaby po prostu żadnej czynności. Druga część wyrażenia warunkowego znajdująca się po słowie else działa dokładnie odwrotnie: najpierw zmiennej tvPower nadawana jest wartość true, dopiero potem zaś następuje wywołanie funkcji. Ponownie dzieje się tak ze względu na konstrukcję funkcji changeTheChannel().

Napiszmy teraz funkcje, które pozwolą wybrać kolejny i poprzedni program TV za pomocą przycisków *Up* oraz *Down* na pilocie.

8. Wybierz 1. klatkę warstwy *Actions* na głównej listwie czasowej i otwórz panel *Actions*. Wprowadź podany niżej skrypt tuż pod linią `tvPower:Boolean = false;`:

```
var numberOfChannels:Number = 6;
```

Ta linia kodu deklaruje zmienną o nazwie `numberOfChannels` i nadaje jej wartość `6`. Zmienną tę wykorzystamy w funkcji (napiszemy ją za chwilę), która służyć będzie do zmiany programu TV na kolejny. Pamiętając, że klip *screen_mc* zawiera graficzne odpowiedniki sześciu programów, nadajemy utworzonej zmiennej wartość `6`, która odzwierciedla ich liczbę. Zmienna `numberOfChannels` będzie zapobiegać zwiększeniu numeru programu na większy niż szósty. Przyjrzyjmy się, jak działa to w praktyce.

9. Dodaj następujący fragment skryptu poniżej już istniejącego:

```
function channelUp() {
  if (currentChannel+1<=numberOfChannels) {
    changeTheChannel(currentChannel+1);
  }
}
```

Funkcja ta nie wymaga parametrów, zaś jej zadaniem jest zwiększanie numeru programu o 1, zawsze wtedy, gdy zostanie ona wywołana. Jednak funkcja zawiera pewien „mechanizm bezpieczeństwa", zapobiegający przekroczeniu dostępnej liczby kanałów (w tym wypadku programu 6.). Przypominamy, że wartość zmiennej o nazwie `currentChannel` zmieniana jest, za każdym razem gdy wywołana jest funkcja `changeTheChannel()` (patrz punkt 3. ćwiczenia). Wartość tej zmiennej odzwierciedla bieżący numer programu minus 1. A zatem, jeśli wyświetlany jest akurat program 4., wartość ta będzie wynosić `3`. Zanim jednak wykonana zostanie funkcja `channelUp()`, uruchamiane jest wyrażenie warunkowe `if` sprawdzające, czy zwiększenie bieżącego numeru programu o 1 (wartość zmiennej `currentChannel + 1`) nadal będzie mniejsza lub co najwyżej równa dostępnej liczbie kanałów (wartość zmiennej `numberOfChannels` czyli `6`). Jeśli warunek ten jest spełniony, następuje wywołanie funkcji `changeTheChannel()` z parametrem o wartości numeru bieżącego kanału zwiększonej o 1. Spowoduje to oczywiście wyświetlenie kolejnego programu. Przedstawione tutaj wyrażenie `if` nie zawiera towarzyszącego wyrażenia `else`: jeśli warunek początkowy nie jest spełniony, to znaczy, że wyświetlany jest akurat program 6. i podejmowanie dalszych działań nie ma sensu.

10. Dodaj kolejny fragment skryptu w bieżącej klatce filmu:

```
function channelDown() {
  if (currentChannel-1>=1) {
    changeTheChannel(currentChannel-1);
  }
}
```

Podobnie jak funkcja `channelUp()`, tak i `channelDown()`do działania nie wymaga podania parametrów. Po wywołaniu sprawdza ona, czy bieżąca wartość zmiennej `currentChannel` *zmniejszona* o 1 byłaby nadal większa lub równa 1, zapobiega jednocześnie zmianie programu na mniejszy niż oznaczony numerem pierwszym. Jeśli warunek ten jest spełniony, następuje wywołanie funkcji `changeTheChannel()` i przekazanie jej bieżącej wartości zmiennej `currentChannel` zmniejszonej o 1. Spowoduje to wyświetlenie poprzedniego w kolejności, w jakiej są ponumerowane, programu. Podobnie jak w przypadku funkcji `channelUp()`, tak i tutaj konstrukcja wyrażenia `if` nie zawiera towarzyszącego wyrażenia `else`. Jeśli warunek początkowy nie jest spełniony, to znaczy, że bieżącym kanałem jest program 1. i nie ma możliwości jego zmiany na „niższy".

Naszedł czas na dodanie wywołań przygotowanych funkcji `channelUp()` i `channelDown()` do przycisków *Up* oraz *Down* na pilocie TV.

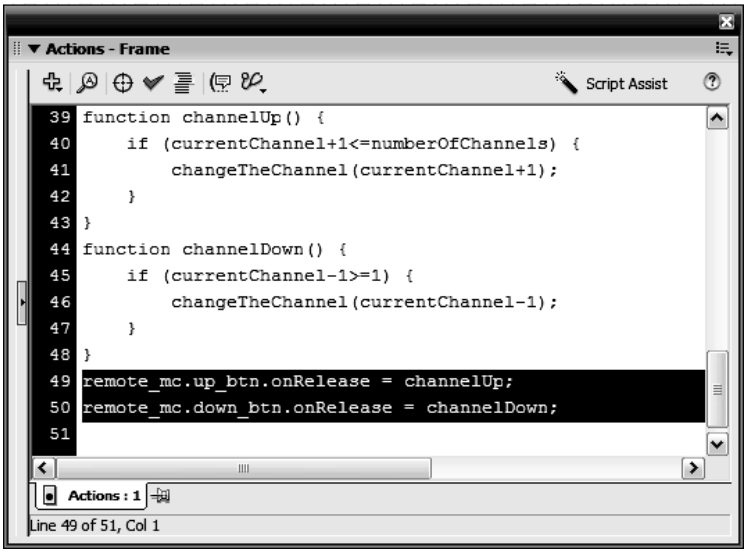
11. Dodaj następującą linię skryptu w 1. klatce filmu:

```
remote_mc.up_btn.onRelease = channelUp;
```

Podana tutaj linia kodu ActionScript przypisuje uchwyt zdarzenia `onRelease` do kopii przycisku *up_btn*, znajdującej się wewnątrz klipu *remote_mc*. Za każdym razem gdy przycisk ten będzie wciśnięty, nastąpi wywołanie funkcji `channelUp()`. Jeżeli numer bieżącego programu zrówna się z górną, dopuszczalną granicą (określoną w zmiennej `numberOfChannels`), wywołanie funkcji `channelUp()` przestanie powodować przełączanie programu na następny.

12. Dodaj następujący skrypt:

```
remote_mc.down_btn.onRelease = channelDown;
```



Podobnie jak w przypadku przycisku *Up* opisanego w 11. punkcie ćwiczenia, skrypt ten przypisuje uchwyt zdarzenia do przycisku *down_btn*, znajdującego się wewnątrz klipu *remote_mc*. Za każdym razem gdy przycisk ten jest wciśnięty, wywoływana jest funkcja `channelDown()`, a wartość zmiennej `currentChannel` jest zmniejszana (tak długo, jak długo będzie ona większa niż dolna, dopuszczalna granica). W wyniku tego na ekranie TV wyświetlany jest odpowiedni program.

13. Wydaj polecenie *Control/Test Movie*. Włącz telewizor, korzystając z przycisku *Power*, a następnie zmień wyświetlany program, posługując się przyciskami *Up* i *Down*.

Zwróć uwagę, że możesz wybrać dowolny program za pomocą przycisków numerycznych, a następnie przełączyć go na poprzedni lub następny przy użyciu przycisków *Up* i *Down*. Stało się to możliwe po wykorzystaniu zmiennej przechowującej bieżący numer programu oraz funkcji niezmiernie upraszczających tego typu zadania.

14. Zakończ testowanie filmu i zapisz projekt w pliku o nazwie *television3.fla*.

Zaprogramowałeś właśnie działanie pilota TV umożliwiającego zmianę wyświetlanych programów. W naszym następnym ćwiczeniu wykorzystamy funkcje w nieco inny niż dotąd sposób, dodając możliwość wyświetlania tekstowego opisu odbieranego właśnie programu.

Zmienne lokalne oraz funkcje, które zwracają określoną wartość

Do zmiennych, które tworzyliśmy i wykorzystywaliśmy do tej pory, można było odwołać się w dowolnym momencie, z poziomu dowolnego skryptu umieszczonego w filmie Flasha. Jednak istnieją również zmienne *lokalne*, które tworzone są i wykorzystywane jedynie w obrębie definicji

określonej funkcji. Innymi słowy, zmienna lokalna tworzona jest wewnątrz definicji funkcji i wykorzystywana przez tę właśnie funkcję po jej wywołaniu, a następnie po zakończeniu działania automatycznie usuwana. Zmienne lokalne istnieją jedynie w obrębie funkcji, w której zostały zadeklarowane.

Choć stosowanie zmiennych lokalnych podczas programowania w języku ActionScript nie jest absolutnie konieczne, posługiwanie się nimi stanowi dobry programistyczny zwyczaj. Aplikacje wymagające wielu skomplikowanych obliczeń wykorzystują mnóstwo zmiennych, które mogą spowolnić jej działanie z upływem czasu. Wykorzystując zmienne lokalne, minimalizujesz wykorzystanie dostępnej pamięci i zapobiegasz występowaniu *konfliktów nazw*, mogących pojawić się w dużych projektach podczas spontanicznego tworzenia nowych zmiennych i nadawania im nazw, które były już wcześniej wykorzystane. Zmienne lokalne zadeklarowane w jednej funkcji mogą mieć identyczne nazwy jak zmienne istniejące w innej — nawet jeśli definicje obu funkcji znajdują się na tej samej listwie czasowej. Jest to możliwe, ponieważ Flash rozpatruje daną zmienną lokalną jedynie w obszarze funkcji, w którym została zadeklarowana.

Jest tylko jedna metoda ręcznego tworzenia zmiennych lokalnych; konstrukcji tej używałeś dotąd z powodzeniem przy tworzeniu zwykłych zmiennych:

```
var myName:String = "Jobe";
```

Zmienna ta staje się lokalna po prostu przez umieszczenie jej deklaracji *wewnątrz* definicji funkcji i zastosowanie konstrukcji ze słowem kluczowym `var`.

Posłużmy się pewnym przykładem, aby lepiej wyjaśnić ideę deklarowania funkcji lokalnych. W naszym poprzednim ćwiczeniu zadeklarowaliśmy (utworzyliśmy) zmienną o nazwie `currentChannel` i umieściliśmy ją w pierwszej klatce głównej listwy czasowej. Do jej utworzenia posłużyła następująca konstrukcja:

```
var currentChannel:Number;
```

Ponieważ podana tutaj linia skryptu zawierająca deklarację zmiennej znajdowała się w 1. klatce głównej listwy czasowej, *nie zaś* wewnątrz definicji funkcji, zmienna ta stanie się dostępna na głównej listwie czasowej. Jeżeli umieścilibyśmy identyczną deklarację wewnątrz definicji jakiejś funkcji, zmienna `currentChannel` byłaby traktowana jako lokalna (przynależna jedynie do tej funkcji): istniałaby wyłącznie po wywołaniu tej funkcji i usuwana po zakończeniu jej działania. Spróbuj potraktować zmienne lokalne jako tymczasowe wartości przeznaczone do wykorzystania jedynie w obrębie wybranej funkcji.

Jeżeli chciałbyś wewnątrz funkcji zadeklarować zmienną, która będzie dostępna z zewnątrz, pomiń w deklaracji słowo kluczowe `var` i skorzystaj z następującej konstrukcji:

```
name = "Jobe";
```

Uwaga

Zmienne ogólnodostępne najlepiej deklarować poza obrębem funkcji. Deklaracja taka jest traktowana jako poprawna, ponieważ ułatwia zgrupowanie wszystkich tego typu zmiennych w jednym miejscu. Porządek i przejrzystość programowania staje się bardzo istotna szczególnie wówczas, gdy chciałbyś powrócić do swojego projektu po kilku miesiącach lub przekazać go innemu programiście.

W jednej linii kodu umieszczonej wewnątrz funkcji możesz zadeklarować wiele zmiennych lokalnych:

```
var firstName:String = "Jobe", lastName:String = "Makar", email:String =
"jobe@electrotank.com";
```

Zwracanie wartości przez funkcję

Funkcje niekoniecznie muszą służyć wyłącznie jako zestaw poleceń przeznaczonych do wykonania; możesz również wykorzystać je w charakterze miniprogramów działających w obrębie projektu, przetwarzających wysyłane im informacje i zwracających określony rezultat. Przyjrzyjmy się następującej definicji funkcji:

```
function buyCD(availableFunds:Number, currentDay:String):Boolean {
  var myVariable:Boolean;
  if(currentDay != "Sunday" && availableFunds >= 20) {
    myVariable = true;
  } else {
    myVariable = false;
  }
  return myVariable;
}
```

Podczas wywołania funkcji wysyłane są do niej dwa parametry — `availableFunds` oraz `currentDay`. Funkcja przetwarza wartości tych parametrów za pomocą wyrażenia warunkowego `if/else`. Efektem jego działania jest przypisanie zmiennej `myVariable` wartości `true` lub `false`. Gdy skorzystamy z wyrażenia `return` (widocznego na końcu definicji funkcji), wartość tej zmiennej zwracana jest do skryptu, który wywołał funkcję. Spróbujmy przeanalizować następujący przykład:

```
var idealCircumstances:Boolean = buyCD(19, "Friday");
if (idealCircumstances == true) {
  gotoAndPlay("Happiness");
} else {
  gotoAndPlay("StayHome");
}
```

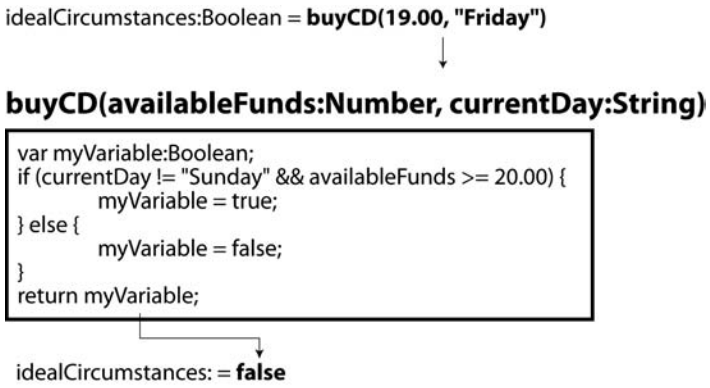
Szczególną uwagę zwróć na następującą linię kodu:

```
var idealCircumstances:Boolean = buyCD(19, "Friday");
```

Po prawej stronie znaku = znajduje się faktyczne wywołanie funkcji, które wysyła wartości `19` oraz `"Friday"` do funkcji `buyCD()`. Pamiętając, w jaki sposób zdefiniowana została funkcja, wiemy, że zostaną one wykorzystane do określenia wartości, jaką przyjmie zmienna `myVariable`. Może to być jedna z dwóch wartości logicznych: `true` lub `false`. Wysłanie określonych parametrów (`19`, `"Friday"`) do funkcji spowoduje, że zmienna `myVariable` będzie miała wartość `false`. Ponieważ ostatnia linia kodu naszej funkcji zakłada zwrot tej wartości za pomocą wyrażenia `return myVariable;`, jest ona zwracana do skryptu, który wywołał funkcję. Otrzymujemy zatem:

```
idealCircumstances = false;
```

W istocie wykorzystaliśmy odwołanie do funkcji, aby błyskawicznie ustalić wartość zmiennej `idealCircumstances`. Po określeniu tej wartości za pomocą zmiennej `idealCircumstances` może ona być wykorzystana w pozostałej części skryptu, co demonstruje przykładowy skrypt.



Wskazówka *Możesz zastosować polecenie return, aby zwrócić dowolny typ danych, włączając wartości zmiennych, tablic i innych obiektów.*

Wiesz już, że funkcje mogą zwracać pewne wartości. Nadszedł zatem odpowiedni moment, aby zwrócić uwagę na drobną zmianę, jakiej poddaliśmy składnię definicji naszej funkcji. Pierwsza linia funkcji `buyCD()` wygląda następująco:

```
function buyCD(availableFunds:Number, currentDay:String):Boolean {
```

Między nawiasem zamykającym sekcję parametrów a otwarciem nawiasu klamrowego na końcu umieściliśmy następujące wyrażenie `:Boolean`. Dodatek ten wskazuje, że funkcja po wywołaniu zawsze zwraca pewną wartość. W tym przypadku jest to jedna z wartości logicznych `true` lub `false`, stąd powód zastosowania typu `:Boolean`. Definicja funkcji zwracającej wartość numeryczną wyglądałaby następująco:

```
function myNumberFunction(param1:Number, param2:Boolean):Number {
  // Akcje
}
```

Definicja funkcji, która zwraca wartość w postaci łańcucha znaków, wyglądałaby np. tak:

```
function myNumberFunction(param1:Number, param2:Boolean):String {
  // Akcje
}
```

Tak wyglądałaby definicja funkcji, która zwraca wartość w postaci tablicy:

```
function myNumberFunction(param1:Number, param2:Boolean):Array {
  // Akcje
}
```

itd.

Jeżeli funkcja nie zwraca żadnej wartości (podobnie jak wszystkie funkcje zdefiniowane do tej pory podczas bieżącej lekcji), konstrukcja jej definicji powinna mieć następującą postać:

```
function myNumberFunction(param1:Number, param2:Boolean):Void {
  // Akcje
}
```

Wyrażenie `:Void` oznacza funkcję, która nie zwraca żadnej wartości.

Choć w funkcjach, którymi posługiwaliśmy się w trakcie bieżącej lekcji, nie korzystaliśmy z takiej konstrukcji definicji (mimo to nadal działają one poprawnie), zastosowanie podanej składni stanowi bardzo poprawny nawyk, a co więcej — powinno przyspieszyć wykonywanie kodu ActionScript. Wzrost prędkości będzie odczuwalny jedynie w przypadku projektu zawierającego wiele funkcji.

W naszym kolejnym ćwiczeniu będziemy korzystać zarówno ze zmiennych lokalnych, jak również funkcji, które zwracają określoną wartość. Zaprogramujemy dekoder telewizji kablowej, znajdujący się pod telewizorem w taki sposób, aby wyświetlał nazwę bieżącego kanału. Utworzymy w tym celu funkcję, która będzie konstruować łańcuch tekstowy przeznaczony do wyświetlenia na dekoderze.

1. Otwórz plik *television3.fla*.

   Projekt ten stanowi kontynuację naszego poprzedniego ćwiczenia. Skupimy się tym razem na kopii klipu o nazwie *cableBox_mc* (przypomina on wyglądem dekoder telewizji satelitarnej lub kablowej). Klip ten zawiera proste elementy graficzne oraz pole tekstowe o nazwie *cableDisplay_txt*. W polu tym wyświetlane będą nazwy różnych programów, w zależności od tego, który z nich zostanie wybrany za pomocą pilota.

2. Wybierz 1. klatkę filmu i, korzystając z panelu *Actions*, wprowadź następujący skrypt poniżej linii `numberOfChannels = 6;`:

```
var channelNames:Array = ["", "News", "Classics", "Family", "Cartoons",
"Horror", "Westerns"];
```

Utworzyłeś właśnie tablicę o nazwie `channelNames`. Zawiera ona nazwy, które będą dynamicznie umieszczane w łańcuchu tekstowym wyświetlanym na dekoderze. Siedem łańcuchów tekstowych, stanowiących elementy tej tablicy, oddzielono przecinkami (pierwszy z nich może wydawać się dość nieczytelny, ponieważ stanowi po prostu łańcuch pusty oznaczony w następujący sposób: `""`). Każdy z tych elementów ma przyporządkowaną pewną liczbę, wyznaczającą jego *pozycję* (*index*) w tablicy. Pierwszy z nich znajduje się na pozycji 0. Przykładowo `channelNames[0] = ""` (pusty), `channelNames[1] = "News"`, `channelNames[2] = "Classics"` itd. Zrozumienie tego zagadnienia ma kluczowe znaczenie podczas realizacji dalszej części ćwiczenia.

Uwaga *Więcej informacji dotyczących tablic znajdziesz w lekcji 4., „Tablice i pętle".*

Utwórzmy funkcję, która wykorzysta elementy tekstowe umieszczone w tej tablicy do wyświetlania komunikatu na dekoderze TV.

3. Wprowadź kolejny fragment skryptu w 1. klatce filmu, poniżej już istniejącego:

```
function displayCableText():String {
  var displayText:String;
  if (currentChannel != 0) {
    displayText = "You are viewing "+channelNames[currentChannel]+".";
  } else {
    displayText = "";
  }
  return displayText;
}
```

Uwaga *Definicja ta mogłaby zostać równie dobrze umieszczona przed, jak i po określeniu uchwytów zdarzeń; jej położenie nie ma większego znaczenia i stanowi kwestię osobistych nawyków i przyzwyczajeń.*

Skrypt ten definiuje funkcję `displayCableText()`, która nie wymaga podania żadnych parametrów. Służy ona do dynamicznego zbudowania łańcucha tekstowego, który pojawi się w polu tekstowym *cableDisplay_txt*, umieszczonym w kopii klipu o nazwie *cableBox_mc*. Funkcja ta zwraca zbudowany łańcuch za pomocą polecenia `return`. Zawiera ona wyrażenie warunkowe sprawdzające, czy bieżący program nie jest przypadkiem oznaczony liczbą 0, co oznacza, że telewizor jest wyłączony. Jeśli warunek ten jest spełniony, tworzona jest zmienna lokalna o nazwie `displayText`, a umieszczany w niej łańcuch tekstowy budowany jest zarówno z elementu tablicy `channelNames`, jak i bieżącej wartości zmiennej `currentChannel`. Jeśli przykładowo wartość tej zmiennej w momencie wywołania funkcji wynosi `4`, konstrukcja łańcucha wygląda następująco:

```
displayText = "You are viewing "+channelNames[4]+".";
```

Ponieważ na pozycji 4. tablicy `channelNames` znajduje się łańcuch tekstowy `Cartoons`, możemy podstawić go do przedstawionego przed chwilą wyrażenia:

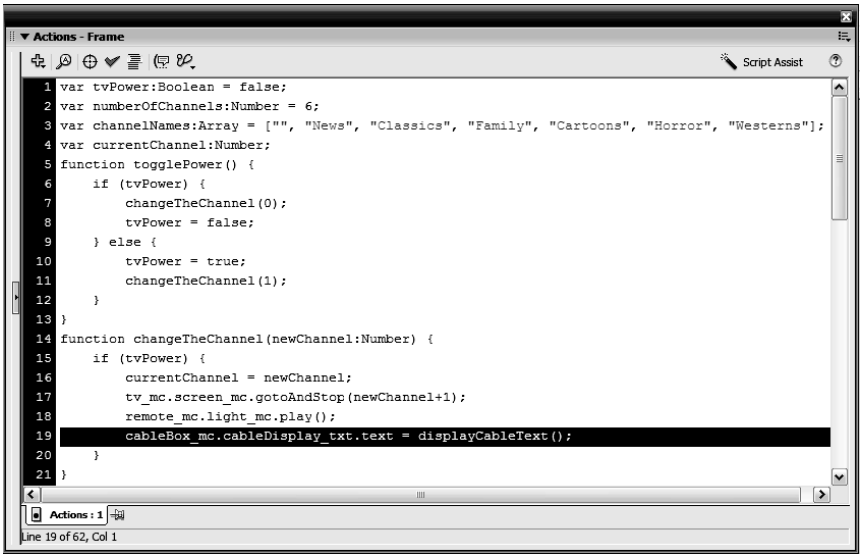
```
displayText = "You are viewing Cartoons.";
```

Jeżeli pierwsza część wyrażenia warunkowego zawartego w funkcji nie jest spełniona (`else`), zmiennej lokalnej `displayText` przypisywany jest łańcuch pusty (`""`). Funkcja kończy się

zwróceniem wartości zmiennej `displayText`. Dokąd jednak zostanie zwrócona ta wartość? Wyjaśnimy to w następnym punkcie ćwiczenia. Ponieważ zmienna `displayText` została zadeklarowana jako lokalna (za pomocą słowa kluczowego `var`), jest ona usuwana z pamięci natychmiast po zwróceniu wartości przez funkcję.

4. Korzystając z panelu *Actions*, zmodyfikuj funkcję `changeTheChannel()` przez dodanie następującego kodu poniżej piątej linii definicji tej funkcji:

```
cableBox_mc.cableDisplay_txt.text = displayCableText();
```



Zmodyfikowaliśmy funkcję `changeTheChannel()` w taki sposób, że za każdym razem przy zmianie kanału i wywołaniu funkcji `changeChannel()` w polu tekstowym *cableDisplay_txt* (znajdującym się wewnątrz klipu filmowego *cableBox_mc*) wyświetlony zostanie odpowiedni tekst. Dodana linia kodu ActionScript ustala wartość kopii pola tekstowego *cableDisplay_txt* (które jest dynamicznym polem tekstowym umieszczonym na naszym dekoderze), korzysta przy tym z wartości zwróconej przez funkcję `displayCableText()`. Funkcja ta zostaje wywołana, zrealizowana, a efekt jej działania umieszczany jest po znaku równości (=). Takie właśnie działanie nosi nazwę *zwracania* wartości przez funkcję. Wartość uzyskana wewnątrz funkcji jest zwracana i umieszczana w linii skryptu, który spowodował jej wywołanie. Jest to również doskonały przykład tego, w jaki sposób stosowanie funkcji pomaga zaoszczędzić mnóstwo czasu — konstrukcję funkcji `changeTheChannel()` zmieniliśmy tylko w jednym miejscu, a mimo to zmiana ta jest widoczna i dostępna przy każdym wywołaniu tej funkcji!

5. Wydaj polecenie *Control/Test Movie*. Włącz telewizor przyciskiem *Power*. Zmień kilkakrotnie wyświetlany program.

Za każdym razem gdy wciśniesz przycisk realizujący zmianę kanału, pole tekstowe umieszczone na dekoderze TV wyświetla jego nazwę. Utworzyłeś prostą aplikację wykorzystującą sześć funkcji realizujących szereg różnorodnych zadań (zobacz rysunek na następnej stronie).

6. Zakończ testowanie filmu i zapisz projekt w pliku *television4.fla*.

Zakończyliśmy pracę nad projektem. Wiedza, którą zdobyłeś, będzie wielokrotnie wykorzystywana w naszych kolejnych lekcjach.

Podsumowanie

W czasie tej lekcji:

- Napisałeś funkcje, używając różnych składni
- Przekazałeś funkcjom argumenty w wywołaniach
- Użyłeś zmiennych lokalnych
- Zwróciłeś wyniki wywołania funkcji i wykorzystałeś je