Pętlami nazywamy konstrukcje języka, które pozwalają na wielokrotne wykonywanie powtarzających się instrukcji. Przykładowo, jeśli trzeba 10 razy wyświetlić na ekranie pewien napis, to można wykorzystać w tym celu 10 instrukcji document.write("napis"), jednak zdecydowanie lepiej i prościej użyć jednej z pętli. Pętle występujące w języku JavaScript możemy podzielić na dwa główne rodzaje:

- pętle typu for (w tym for i for...in)
- pętle typu while (w tym while i do...while)

Pętla for

Pętla typu for ma ogólną postać:

```
for(wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące){

  instrukcje do wykonania

}
```

wyrażenie początkowe jest stosowane do zainicjalizowania zmiennej używanej jako licznik liczby wykonań pętli. wyrażenie warunkowe określa warunek, jaki musi być spełniony, aby dokonać kolejnego przejścia w pętli, natomiast wyrażenie modyfikujące jest zwykle używane do modyfikacji zmiennej będącej licznikiem. Sposób działania takiej pętli najłatwiej pokazać na konkretnym przykładzie. Został on przedstawiony na listingu 2.13.

Listing 2.13. Ilustracja działania pętli for

```
<script type="text/javascript">

for(i = 0; i < 10; i++){

  document.write(i + " ");

}

</script>
```

Taką konstrukcję należy rozumieć następująco: utwórz zmienną i i przypisz jej wartość zero (i = 0), następnie, dopóki wartość i jest mniejsza od 10 (i < 10), wykonuj instrukcje znajdujące się wewnątrz pętli (instrukcja document.write) oraz zwiększaj i o jeden (i++). Tym samym na ekranie (rysunek 2.4) pojawi się ciąg liczb od 0 do 9 odzwierciedlających kolejne stany zmiennej i, którą nazywamy zmienną iteracyjną, czyli kontrolującą kolejne przebiegi (iteracje) pętli.

Rysunek 2.4. Efekt działania pętli

Pętla for...in

Pętla for...in pozwala na przejrzenie wszystkich właściwości obiektu bądź indeksów tablicy. Te tematy zostaną omówione w dalszej części kursu, wtedy też zostaną podane przykłady jej wykorzystania. Ogólna struktura tej pętli jest następująca:

```
for(właściwość in obiekt){

  //instrukcje do wykonania

}
```

W każdym jej przebiegu pod właściwość będzie podstawiana kolejna właściwość obiektu wskazywanego przez obiekt. **Pętla while**

Pętla typu while służy, podobnie jak for, do wykonywania powtarzających się czynności. Pętlę for najczęściej wykorzystuje się, kiedy liczba powtarzanych operacji jest znana, natomiast pętlę while, kiedy liczby powtórzeń nie znamy, a zakończenie pętli jest uzależnione od spełnienia pewnego warunku. Taki podział jest jednak dosyć umowny, gdyż oba typy pętli można zapisać w taki sposób, aby były swoimi funkcjonalnymi odpowiednikami. Ogólna postać pętli while wygląda następująco:

```
while (warunek){

  instrukcje;

}
```

i oznacza: dopóki warunek jest prawdziwy, wykonuj instrukcje. Wykorzystajmy więc pętlę while do wyświetlenia powtarzającego się napisu (listing 2.14).

Listing 2.14. Ilustracja działania pętli typu while

```
<script type="text/javascript">

var i = 0;

while(i < 10){

  document.write("Pętla while [i = " + i + "]");

  document.write("<br />");

  i++;

}

</script>
```

W tym przypadku instrukcje znajdujące się wewnątrz pętli wykonywane są dopóty, dopóki wartość zmiennej i jest

mniejsza od 10, warunkiem zakończenia pętli jest bowiem i < 10. Ponieważ początkową wartością i jest 0, a wewnątrz pętli znajduje się instrukcja zwiększająca w każdym jej przebiegu wartość i o 1 (i++), mamy pewność, że pętla się zakończy. Na warunek zakończenia trzeba zwracać szczególną uwagę w przypadku pętli while, gdyż łatwo napisać pętlę wykonującą się w nieskończoność.

Zmodyfikujmy teraz powyższy przykład w taki sposób, aby modyfikacja zmiennej i odbywała się w wyrażeniu warunkowym, a nie we wnętrzu pętli (listing 2.15).

Listing 2.15. Modyfikacja zmiennej interacyjnej w wyrażeniu warunkowym

```
<script type="text/javascript">

var i = 0;

while(i++ < 10){

  document.write("Pętla while [i = " + i + "]");

  document.write("<br />");

}

</script>
```

Jak widać, nie było to bardzo skomplikowane zadanie. Zwróćmy jednak uwagę, że nie uzyskaliśmy w ten sposób w pełni funkcjonalnego odpowiednika poprzedniego skryptu. Co prawda napis zostanie wyświetlony tak jak w poprzednim przypadku 10 razy, ale wartość zmiennej wewnątrz pętli zmieniać się będzie od 1 do 10, a nie od 0 do 9! Byłaby to bardzo poważna zmiana, gdyby zmienna i była wykorzystywana do jakichś czynności, np. do indeksowania tablicy. **Pętla do...while**

Kolejny rodzaj pętli występującej w JavaScripcie to do...while. Jak łatwo się domyślić, jest ona blisko spokrewniona ze zwykłą pętlą while. Jej ogólna postać jest następująca:

```
do{

  instrukcje;

}

while(warunek);
```

Co oznacza: wykonuj instrukcje, dopóki warunek jest prawdziwy. A zatem spróbujmy wyświetlić 10 napisów za pomocą tego rodzaju pętli. Jak to zrobić, zostało zilustrowane na listingu 2.16.

Listing 2.16. Ilustracja działania pętli typu do…while

```
<script type="text/javascript">

var i = 0;

do{

  document.write("Pętla do...while [i = " + i + "]");
```

```
  document.write("<br />");

}

while(i++ < 9);

</script>
```

Główna różnica w stosunku do pętli while jest taka, że w tym przypadku najpierw są wykonywane instrukcje, a dopiero potem jest sprawdzany warunek. Wynika z tego, że instrukcje z wnętrza pętli while są wykonywane zawsze przynajmniej jeden raz, nawet jeśli warunek będzie fałszywy. Zwróćmy również uwagę, że w związku z odmienną konstrukcją pętli nieco inaczej wygląda wyrażenie warunkowe (i++ < 9). Tym razem jest sprawdzane, czy i jest mniejsze od 9. Gdyby pozostawić taki sam warunek, jak w przypadku pętli while (i++ < 10), instrukcje document.write zostałyby wykonane 11 razy.

O tym, że pętla do…while jest wykonywana co najmniej raz, nawet wtedy, gdy warunek jej kontynuacji jest ewidentnie fałszywy, można się przekonać, uruchamiając kod widoczny na listingu 2.17.

Listing 2.17. Pętla do…while z zawsze fałszywym warunkiem

```
<script type="text/javascript">

var i = 0;

do{

  document.write("Wnętrze pętli for");

}

while(false);

</script>
```

Wyrażenie warunkowe w tej postaci jest ewidentnie fałszywe — to po prostu wartość logiczna false. Mimo to po wykonaniu powyższego kodu na ekranie pojawi się jeden napis Wnętrze pęli for. Jest to najlepszy dowód na to, że warunek jest sprawdzany nie przed, ale po każdym przebiegu pętli. **Przerywanie pętli**

Działanie każdej z pętli może być przerwane w dowolnym momencie za pomocą instrukcji break (break z ang. znaczy przerywać), tej samej, która służyła do opuszczenia bloku switch. Jeśli zatem break pojawi się wewnątrz pętli, zakończy ona swoje działanie. Najczęściej instrukcja ta jest stosowana po sprawdzeniu jakiegoś warunku. Przykład zastosowania break został przedstawiony na listingu 2.18.

Listing 2.18. Przykład użycia instrukcji break

```
<script type="text/javascript">

var i = 0;

while(true){

  document.write("napis [i = " + i + "] <br />");

  if(i++ >= 9) break;
```

```
}

</script>
```

W kodzie została umieszczona specyficzna pętla while, powodująca wyświetlenie dziesięciu napisów (w każdym z nich jest podawany stan zmiennej i). Zwróćmy jednak uwagę na warunek pętli — jest to true, a zatem warunek ten jest zawsze prawdziwy (analogicznie do przykładu z poprzedniego listingu, gdzie warunek był zawsze fałszywy). Oczywiście taka pętla wykonywałaby się w nieskończoność, gdyby nie znajdująca się wewnątrz instrukcja break. Z kolei warunkiem wykonania break jest i >= 9, czyli przerwanie pętli następuje wtedy, kiedy zmienna i osiągnie co najmniej wartość 9. Ponieważ i jest zwiększane o jeden w każdym przebiegu pętli (i++), a jego początkowa wartość to 0, mamy pewność, że pętla zostanie wykonana 10 razy, po czym zostanie przerwana (warunek mógłby mieć zatem również postać i++ == 9). **Kontynuacja pętli**

O ile instrukcja break powodowała przerwanie wykonywania pętli oraz jej opuszczenie, instrukcja continue powoduje przejście do jej kolejnej iteracji. Jeśli zatem wewnątrz pętli znajdzie się instrukcja continue, bieżąca iteracja (przebieg) zostanie przerwana oraz rozpocznie się kolejna (chyba że bieżąca iteracja była ostatnią). Zobaczmy, jak to działa na konkretnym przykładzie, który został przedstawiony na listingu 2.19.

Listing 2.19. Przykład użycia instrukcji continue

```
<script type="text/javascript">

for(var i = 1; i <= 20; i++){

  if(i % 2 != 0) continue;

  document.write(i + " ");

}

</script>
```

Jest to pętla for, która wyświetla liczby całkowite z zakresu 1 – 20 podzielne przez 2, tak jak jest to widoczne na rysunku 2.5.

Rysunek 2.5. Wyświetlenie parzystych liczb z danego zakresu

Wewnątrz pętli znajduje się instrukcja warunkowa, sprawdzająca warunek i % 2 != 0, czyli badająca, czy reszta z dzielenia i przez 2 jest równa zero (znak % oznacza działanie polegające na obliczeniu reszty z dzielenie, czyli tzw. dzielenie modulo). Jeśli ten warunek jest fałszywy (reszta jest różna od zera), oznacza to, że wartość zapisana w i nie jest podzielna przez 2, jest zatem wykonywana instrukcja continue. Jak już wiemy, powoduje ona rozpoczęcie kolejnej iteracji pętli, czyli zwiększenie wartości zmiennej i o jeden i przejście na początek pętli (do pierwszej instrukcji). Tym samym, jeśli wartość i jest niepodzielna przez dwa, nie zostanie wykonana znajdująca się za warunkiem instrukcja document.write(i + " ");, dana wartość nie pojawi się więc na ekranie, a to właśnie było naszym celem.

Rzecz jasna, zadanie to można wykonać bez użycia instrukcji continue, np. tak, jak pokazano na listingu 2.20, ale bardzo dobrze ilustruje ono istotę jej działania.

Listing 2.20. Wyświetlanie liczb parzystych bez użycia instrukcji continue

```
<script type="text/javascript">

for(var i = 1; i <= 20; i++){

  if(i % 2 == 0)

    document.write(i + " ");

}

</script>
```

Zagnieżdżanie pętli

Wewnątrz każdej pętli można umieścić dowolne instrukcje, również kolejną pętlę. Mogą być więc one zagnieżdżane. Najczęściej zagnieżdżane są pętle for, taka konstrukcja ma następującą postać:

```
for(wyrażenie początkowe 1; wyrażenie warunkowe 1; wyrażenie modyfikujące 1){

  for(wyrażenie początkowe 2; wyrażenie warunkowe 2; wyrażenie modyfikujące 2){

    instrukcje do wykonania

  }

}
```

W takiej sytuacji w każdym przebiegu pętli zewnętrznej są wykonywane wszystkie przebiegi pętli wewnętrznej. Obrazuje to przykład widoczny na listingu 2.21.

Listing 2.21. Zagnieżdżanie pętli typu for

```
<script type="text/javascript">

for(var i = 0; i < 3; i++){

  for(var j = 0; j < 3; j++){

    document.write(i+ " " + j);

    document.write("<br />");
```

```
  }

  document.write("----");

  document.write("<br />");

}

</script>
```

W pętli zewnętrznej zmienną iteracyjną jest i, natomiast w wewnętrznej — j. W pętli wewnętrznej znajduje się instrukcja document.write(i+ " " + j);, która wyświetla aktualny stan tych zmiennych. W pierwszej iteracji zewnętrznej i jest stała, równa 0, a j zmienia się od 0 do 2. W drugiej iteracji zewnętrznej i jest stała, równa 1, a j zmienia się od 0 do 2, natomiast w trzeciej iteracji zewnętrznej i jest stała, równa 1, a j zmienia się od 0 do 2. Zatem wynikiem będą dwie kolumny liczb, z których pierwsza obrazuje kolejne stany zmiennej i, a druga — j.

W podobny sposób można zagnieżdżać pętle while, istnieje również możliwość mieszania typów zagnieżdżonych pętli, tzn. w pętli while może być zagnieżdżona pętla for i odwrotnie, w pętli for może być zagnieżdżona pętla while. Przykładowo: aby wyświetlić dwie kolumny liczb takie jak w poprzednim przykładzie przy użyciu dwóch zagnieżdżonych pętli while, można byłoby użyć kodu z listingu 2.22.

Listing 2.22. Zagnieżdżanie pętli typu while

```
<script type="text/javascript">

var i = 0;

var j = 0;

while(i < 3){

  while(j < 3){

    document.write(i+ " " + j);

    document.write("<br />");

    j++;

  }

  document.write("----");

  document.write("<br />");

  j = 0;

  i++;

}

</script>
```

Zmienne zostały zadeklarowane i zainicjowane na początku kodu, jeszcze przed pętlami. Instrukcje zwiększające wartości zmiennych są natomiast wykonywane wewnątrz pętli. Zmienna j jest zwiększana w pętli wewnętrznej, a

zmienna i — w pętli zewnętrznej. Należy zwrócić uwagę, że przy takiej realizacji zmienna j musi być zerowana po każdym wykonaniu wszystkich iteracji pętli wewnętrznej. Jest to wykonywane przez instrukcję:

```
j = 0;
```

Pominięcie tej instrukcji spowodowałoby, że pętla wewnętrzna zostałaby wykonana tylko raz, bowiem po jej pierwszym wykonaniu j będzie miało wartość 3. To z kolei oznacza, że nie będzie spełniony warunek j < 3, którego spełnienie jest konieczne, aby mogły być wykonane instrukcje wnętrza pętli.

Powyższe pętle while można też zapisać w taki sposób, aby modyfikacje zmiennych sterujących były połączone z instrukcjami warunkowymi określającymi, kiedy pętle mają być kontynuowane. Wtedy kod przyjąłby postać przedstawioną na listingu 2.23.

Listing 2.23. Modyfikacja zmiennych iterujących w wyrażeniach warunkowych pętli

```
<script type="text/javascript">

var i = -1;

var j = -1;

while(i++ < 2){

  while(j++ < 2){

    document.write(i+ " " + j);

    document.write("<br />");

  }

  document.write("----");

  document.write("<br />");

  j = -1;

}

</script>
```

Ogólna struktura kodu w stosunku do poprzedniego przykładu pozostała bez zmian, jednak w szczegółach ta realizacja różni się dosyć znacznie. Zmienne kontrolujące przebieg pętli są deklarowane na początku kodu, ale przypisywana jest im inna wartość. Jest to –1. Musi tak być, ponieważ tym razem zmienne iteracyjne są zwiększane już w wyrażeniach warunkowych i++ < 2 i j++ < 2. To oznacza, że zostaną zwiększone jeszcze przed wykonaniem instrukcji z wnętrza pętli. Tym samym, aby pierwszą wyświetlaną wartością i i j było 0, początkowo muszą być one równe –1.

Z tego samego powodu zmieniły się postacie warunków. Tym razem pętle są kontynuowane, gdy i i j są mniejsze od 2 (w poprzednim przykładzie warunki miały postać i < 3 i j < 3). Musi tak być, ponieważ obie zmienne są najpierw zwiększane, a dopiero potem są sprawdzane warunki. W poprzednim przykładzie było odwrotnie — najpierw były sprawdzane warunki kontynuowania pętli, a dopiero potem (we wnętrzu pętli) zmienne były zwiększane.