Pętle w JS

Pętle (*ang. loop*) są jednym z najważniejszych narzędzi w programowaniu – są to konstrukcje językowe umożliwiające wielokrotne wykonanie zadanego kodu. Bez nich trudno wyobrazić sobie jakąkolwiek aplikację – od prostych obliczeń po zaawansowane algorytmy, gdzie pętle tak często znajdują zastosowanie.

To dzięki pętlom możemy wypisać w sklepie internetowym listę kilkudziesięciu produktów wyjętych z bazy danych, na stronie głównej bloga pokazać ostatnie 20 wpisów z danej kategorii albo w rankingu graczy przedstawić tabelę top 100 serwera.

W JavaScript istnieją trzy podstawowe pętle:

- `for` – stosowana najczęściej gdy dokładnie wiemy, ile razy chcemy powtórzyć operacje.
- `while` – gdy liczba powtórzeń zależy od spełnienia warunku – dopóki jest spełniony, to pętla powtarza działanie.
- `do..while` – analogiczna do `while` bo również warunkowa, lecz z tą różnicą, iż kod wewnątrz pętli na pewno wykona się przynajmniej jeden raz.

Pętla for – gdy znamy liczbę iteracji

Najczęściej używaną, klasyczną pętlą w JavaScript jest

`for`, ponieważ pozwala precyzyjnie określić liczbę powtórzeń. Rozważmy pętlę wypisującą liczby od `1` do `10`:

```
for (var i = 1; i <= 10; i++) {
    console.log("To jest iteracja nr " + i);
}
```

Iteracja to pojęcie oznaczające jedno wykonanie pętli. Iterator zaś to pomocnicza zmienna

`i`. Ponadto warto znać jeszcze nazwy dwóch operacji często wykonywanych w pętlach:

- Inkrementacja to inaczej zwiększenie wartości o dokładnie `1`.
- Dekrementacja to inaczej zmniejszenie wartości o dokładnie `1`.

W powyższej pętli warto zwrócić uwagę na:

- Pierwszy wpis w nagłówku to `var i = 1` – definiujemy tutaj rozpoczęcie powtarzania operacji w pętli, iterator `i` przyjmuje wartość startową `1`.
- Warunek `i <= 10` oznacza, iż pętla będzie działać (wykonywać iteracje), dopóki ten warunek jest spełniony. Jeśli choć raz nie będzie on prawdziwy, to pętla zakończy swoje działanie.
- Inkrementacja `i++` – po każdym powtórzeniu (czyli po każdej iteracji) zwiększaj `i` o dokładnie `1`.

Zliczanie w dół

Rozważmy jeszcze przykład pętli zliczającej w dół, czyli dokonującej dekrementacji od wartości

`10` do `1` (niczym sylwestrowe odliczanie do końca roku):

```
for (var i = 10; i >= 0; i--) {
    console.log("Odliczanie: " + i);
```

```
}
```

Warunek tym razem zmienił znak:

`i >= 0` gdyż powtarzanie ma się odbywać dopóki wartość pozostaje większa lub równa zero. Wstawienie niewłaściwego znaku w warunku – na przykład: `i <= 10` spowodowałoby stworzenie nieskończonej pętli! W końcu wartości maleją od początkowego `10` w każdej iteracji o dokładnie jeden (`10` , `9` , `8` , `7` , `6` , `5` , …) co sprawia, że wartości `i` zawsze pozostaną mniejsze lub równe `10` – stworzylibyśmy wówczas pętlę "kręcącą się" w nieskończoność!

Zmiana kroku iteracji

Oczywiście pętle nie muszą zmieniać wartości

`i` zawsze o wartość jeden – możemy np. zwiększać iterator o 2 w każdym wykonaniu:

```
for (var i = 2; i <= 10; i += 2) {
    console.log(i);
}
```

Rezultatem wykonania takiej pętli będzie w logu konsoli ciąg kolejnych liczb parzystych:

`2` `4` `6` `8` `10` . W żargonie programistów, liczbę o którą w każdej iteracji zmieniamy wartość `i` nazywamy krokiem pętli (*ang. step*).

Pętla while – iterujemy dopóki warunek jest spełniony

Jeśli nie wiemy z góry, ile dokładnie razy kod ma się wykonać, lepiej użyć pętli sterowanej warunkiem – jest to równie popularna pętla

`while` . Działa to w sposób bardzo naturalny: *Wykonuj zadany blok kodu tak długo, jak warunek pętli zwraca* `true` (tzw. warunek trwania pętli).

Rozważmy prostą grę w stylu "zgadnij liczbę z przedziału od 1 do 100":

```
var tajemnica = 42;
var zgadnij = 0;
while (zgadnij != tajemnica) {
    zgadnij = Number(prompt("Zgadnij liczbę od 1 do 100:"));
    if (zgadnij < tajemnica) {
        alert("Za mało!");
    } else if (zgadnij > tajemnica) {
        alert("Za dużo!");
    }
}
alert("Brawo! Wygrywasz!");
```

Pętla

`while` będzie się wykonywać, dopóki nasza odpowiedź – wartość w zmiennej `zgadnij` będzie inna aniżeli wartość "tajemnicy" do odgadnięcia równa `42` . Za każdym razem prosimy użytkownika o nową liczbę z klawiatury i sprawdzamy, czy jest to za mało, za dużo, czy równo w porównaniu do odgadywanej wartości. Jeśli więc wpiszemy

`42` to przerwiemy pętlę (ponieważ warunek `(zgadnij != tajemnica)` nie jest już prawdziwy) i wyświetlimy gratulacje.

Reasumując: pętla

`while` doskonale sprawdza się w sytuacjach, gdy **nie wiemy z góry, ile razy będziemy musieli coś powtarzać**. W grze powyżej nie wiemy, ile razy użytkownik będzie zgadywać – dlatego właśnie `while` jest tutaj idealnym wyborem.

Pętla do..while – co najmniej jedno wykonanie

W JavaScript, oprócz klasycznej pętli

`while`, dysponujemy także pokrewną konstrukcją językową – pętlą `do..while`. Różni się ona jednym ważnym szczegółem: kod w bloku `do` wykona się zawsze przynajmniej raz, nawet jeśli warunek od początku będzie fałszywy! Dzieje się tak dlatego, że warunek trwania pętli sprawdzany jest na końcu pętli, czyli już po wykonaniu minimum jednej iteracji:

```
var haslo = "";
do {
  haslo = prompt("Podaj hasło dostępu:");
} while (haslo != "T4jn3_ha5l0_@dm1n");
alert("Dostęp przyznany!");
```

Dopóki hasło będzie różne od wzorca, pętla będzie powtarzana. Natomiast kluczowym aspektem działania tej pętli jest pewność, iż zawsze przynajmniej raz zapytamy o hasło dostępowe funkcją

`prompt()`.

Przerywanie lub pomijanie iteracji – break i continue

Podczas implementowania pętli czasami pojawia się potrzeba **przerwania działania całej pętli** lub **pominięcia tylko konkretnej iteracji**, kiedy zajdą określone warunki. W takich sytuacjach JavaScript oferuje nam dwa bardzo przydatne słowa kluczowe:

- `break` (*ang. zerwij, przerwij*) – natychmiast przerywa działanie całej pętli.
- `continue` (*ang. kontynuuj*) – pomija tylko wykonanie aktualnej iteracji, ale kontynuuje działanie pętli od kolejnej.

Rozważmy przykład znalezienia w przedziale liczbowym

`(30; 50)` pierwszej liczby podzielnej przez `7`:

```
for (var i = 30; i <= 50; i++) {
    if (i % 7 == 0) {
        alert("Pierwsza liczba podzielna przez 7 to: " + i);
        break;
    }
}
alert("Koniec pętli!");
```

Po napotkaniu w pętli wartości

`35` (pierwsza liczba w tym przedziale podzielna przez `7`) cała pętla zostanie przerwana. Pora zatem także na przykład zastosowania `continue`. Załóżmy, że mamy tablicę imion uczniów, ale z jakiegoś powodu w tablicy pojawiły się puste ciągi (`""`). Chcemy wypisać tylko te imiona, które nie są puste – a więc pomijamy te, które są niepoprawne.

```
var uczniowie = ["Ania", "", "Bartek", "", "", "Jacek"];
for (var i = 0; i < uczniowie.length; i++) {
  if (uczniowie[i] == "") {
    continue; // przerywamy bieżącą iteracje by rozpocząć następną
  }
  alert("Uczeń: " + uczniowie[i]);
}
```

Przechodzimy przez każdy element tablicy

`uczniowie`, a liczbę jej elementów znamy dzięki odczytaniu długości tej kolekcji: `uczniowie.length`. Jeśli natrafiamy w szufladzie tablicy na pusty ciąg znakowy, to dzięki `continue` pominiemy tę bieżącą iterację nie uruchamiając alerta, który w pętli znajduje się przecież poniżej `continue`.

Iterowanie po elementach tablicy – forEach()

W programowaniu często wykonujemy operacje na elementach tablicy – np. chcemy każdy element wypisać, odczytać, sprawdzić czy przekształcić. Zamiast używać na tablicach klasycznych pętli

`for`, `while` lub `do..while`, możemy też skorzystać z bardziej czytelnej metody o nazwie `forEach()` (*ang. dla każdego elementu*):

```
var liczby = [10, 20, 30, 40];
var suma = 0;
liczby.forEach(liczba => {
    suma += liczba;
});
alert("Suma liczb w tablicy wynosi: " + suma);
```

W powyższym przykładzie odczytujemy wszystkie wartości zgromadzone w szufladach tablicy celem obliczenia ich sumy. Co ważne – należy wymyślić roboczą, podręczną nazwę pojedynczej wartości wyjętej z kolekcji – tutaj była to pomocnicza zmienna o nazwie

`liczba`.

W metodzie

`forEach()` możemy także bardzo łatwo uzyskać indeks aktualnie przetwarzanej szuflady tablicy – wystarczy dodać drugi parametr w funkcji:

```
var liczby = [10, 20, 30, 40];
var suma = 0;
liczby.forEach((liczba, indeks) => {
    alert("Dodaję teraz liczbę o indeksie: " + indeks);
    suma += liczba;
});
alert("Suma liczb w tablicy wynosi: " + suma);
```

W niektórych obliczeniach dostęp do indeksu może okazać się konieczny, stąd warto wiedzieć że także w metodzie

`forEach()` mamy do niego bezproblemowy dostęp.

Podsumowanie – którą pętlę wybrać?

Podsumujmy zdobyte w tej lekcji informacje – pora wymienić poznane konstrukcje językowe i odpowiedzieć na pytanie kiedy użyć każdej z nich:

- `for` – kiedy znamy z góry dokładną liczbę powtórzeń.
- `while` – kiedy liczba iteracji zależy od warunku.
- `do..while` – gdy na pewno musimy wykonać kod w pętli przynajmniej raz.
- `break` – gdy chcemy przerwać całą pętlę.
- `continue` – gdy chcemy pominąć bieżącą iterację.
- `forEach()` – metoda iterowania po szufladach tablicy.

Pętle są kluczowym elementem egzaminu INF.03 i przydają się w wielu kodach źródłowych. Warto dobrze je opanować i umieć zastosować w różnych sytuacjach problemowych!