Pętla typu for

Jednym z najczęściej stosowanych typów pętli jest instrukcja for.

```
	for (zainicjowanie_zmiennych;  warunek_kończący_wykonywanie_pętli;
zmiana_zmiennych) {
		kod który zostanie wykonany pewną ilość razy
	}
```

Pętle takie najczęściej stosuje się w sytuacjach, kiedy dokładne znamy liczbę powtórzeń - od - do:

```
	//pętla od 0 do 99
	for (let i=0; i<100; i++) {
		document.write("Nie będę rozmawiał na lekcji Informatyki.");
	}
```

W każdej pętli mamy dostęp do jej licznika:

```
	for (let i=0; i<10; i++) {
		document.write("Wykonanie pętli ", i);
	}


	let sum = 0;

	for (let i=0; i<10; i++) {
		sum += i;
	}

	document.write(sum); //45
```

Pętle spokojnie mogą odliczać w przeciwnym kierunku:

```
for (let i=10; i>0; i--) {
		document.write("Trwa odliczanie", i);
	}

	document.write(str);
```

A sam warunek kończący wcale nie musi wyglądać jak powyżej:

```
const a = 10;
const b = 20;

for (let i=1; i<=a && i<=b; i++) {
        //bo oba muszą być prawdziwe
        document.write("Wypisze się tyle co ma mniejsza liczba", i);
}
```

Jeżeli nie potrzeba, to możemy pominąć dowolną z trzech składowych pętli.
Nie używasz licznika?
Nie deklarujemy zmiennych.

Nie potrzebujesz kończącego warunku?
Nie twórz go.

```
let a = 10;
let i = 0;
for (; i<10 ;) {
        document.write(i);
        i++;
}

for (let j=0; j<10; ) {
        document.write(j);
        j++;
}
```

Tak naprawdę nic nie musimy tutaj podawać, a sama pętla może mieć postać:

```
for (;;) {

}
```

Widziałem to tylko w jednym miejscu w Internecie - w dokumentacji MDN.
Pętla taka staje się nieskończoną pętlą.
Na co dzień polecam jednak podawać wszystkie składowe.
Czytelność i jasność kodu znacząco się powiększa.

Czasami w Internecie spotkasz zapis pętli bez użycia słowa kluczowego var/let:

```
for (i=0; i<10; i++) {...}
```

Zadziałać zadziała, ale nie zalecam chodzić na skróty. Gdy nie używamy słowa kluczowego (var / let) działamy na zmiennej globalnej, co może powodować błędy:

```
var i = "kot";

function test() {
   for (i=0; i<10; i++) {
      document.write(i);
   }
}
test();

document.write(i); //10
```

a i w niektórych sytuacjach może jeszcze bardziej być problematyczne. Dlatego uczmy się pisać ładny kod...

Pętla typu while

Pętla while (dopóki) to kolejny typ pętli, który można spotkać w wielu językach. Struktura tej pętli ma następującą postać:

```
while (wyrażenie_sprawdzające_zakończenie_pętli) {
        ...fragment kodu który będzie powtarzany...
}
```

Zauważ, że w pętli tego typu nie definiujemy ani początkowego licznika, ani nie definiujemy zmiany licznika. Musimy te rzeczy zrobić ręcznie:

```
let i = 1;

while (i <= 100) {
        document.write("Nie będę...");
        i++;
}
```

Jeżeli w powyższym kodzie pętli nie zwiększalibyśmy zmiennej i, wówczas pętla ta wykonywała by się w nieskończoność (infinite loop), co zaowocowało by "zawieszeniem" strony.

Pętlę while zazwyczaj stosuje się w sytuacjach, kiedy nie wiemy dokładnie, ile iteracji (powtórzeń) ma się wykonać. Wyobraź sobie, że chcesz wygenerować unikalny numer ID albo jakąś liczbę. Generujesz więc daną rzecz, następnie sprawdzasz czy wynik pasuje do założeń. Jak nie pasuje, generujesz dalej. I tak do skutku aż trafisz.

```
let i = 0;

while (i < 0.5) {
        document.write(i);
        i = Math.random();
}

document.write(i);
```

Istnieje też nieco inny wariant pętli while:

```
let i = 0;

do {
        i++;
        document.write(i);
} while (i < 5);
```

Taki typ pętli wykona się minimum 1 raz.

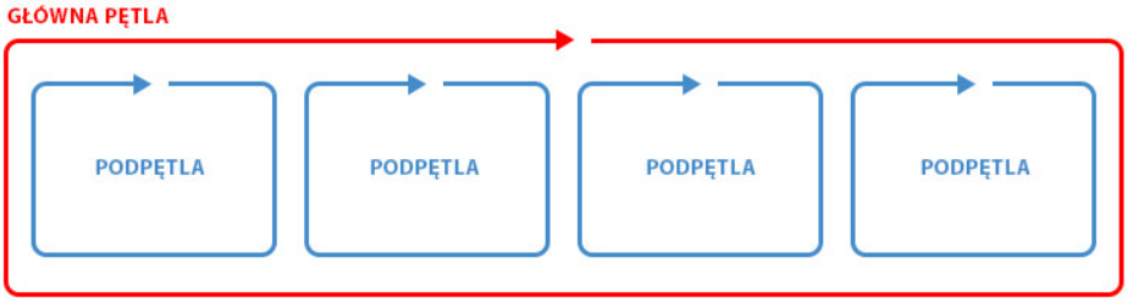
```
let i = 0;

do {
        i++;
        document.write(i);
} while (false); //warunek od początku nie spełniony ale i tak 1 raz się wykona
```

Pętla w pętli

Czasami musimy wykonać zadania "n - wymiarowe".
Dla przykładu przy wypisywaniu tabliczki mnożenia musimy utworzyć 10 kolumn z 10 komórkami. Do takich działań stosujemy pętle w pętlach.

Popatrz na powyższe przykłady. Pętla zaczyna się "kręcić", wykonując swój wewnętrzny kod. W powyższych przykładach to tylko jedna linijka document.write(). Ale przecież takich linii kodu, które mają się powtarzać może być dowolnie wiele. Co więcej - wśród tych linii mogą być zagnieżdżone pętle, które przy każdym przebiegu głównej pętli wykonają swój kod naście razy.




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

        document.write("%c Główna pętla nr: " + i, "color:red");

        for (let j=0; j<6; j++) {
                document.write("%c Pętla wewnętrzna nr: " + j, "color:blue");
        }
}
```

Jak widzisz główna pętla zaczyna działać wykonując po kolei kolejne linie kodu. Wypisuje więc w konsoli jedną czerwoną linijkę. Kod dochodzi do wewnętrznej pętli. Ta zaczyna działać.

Po jej zakończeniu kończy się działanie jednej iteracji głównej pętli. Rozpoczyna się druga iteracja...

Co ważne, wewnętrzne pętle mają dostęp do liczników pętli zewnętrznych.

Bardzo ważne jest by pętle wewnętrzne miały inny licznik niż główna pętla.
W przypadku stosowania zmiennych **var** użycie takiego samego licznika praktycznie zawsze oznacza zawieszenie strony.
W przypadku **let** można niby użyć licznik o takiej samej nazwie, ale tracimy wtedy dodatkowe możliwości:

```
for (let i=0; i<10; i++) {
        for (let j=0; j<6; j++) {
                document.write(`Główna pętla: ${i}, pętla zagnieżdżona: ${j}`)
        }
}
```

```
for (let i=0; i<10; i++) {
	for (let i=0; i<6; i++) {
		//nie mam dostępu do zewnętrznego licznika...
		document.write(`Pętla zagnieżdżona: ${i}`);
	}
}
```

Przykłady użycia pętli

Powiedzmy, że chcemy w konsoli wypisać tekst:

******

Wykorzystajmy do tego pętlę:

```
let str = "";

for (let j=0; j<6; j++) {
	str += "*";
}

document.write(str);
```

Dodajemy kolejny poziom skomplikowania. Załóżmy, że chcemy wypisać w konsoli poniższy tekst:

******
******
******
******

Użyjmy do tego pętli w pętli:

```
let str = "";

for (let i=0; i<4; i++) {
	for (let j=0; j<6; j++) {
		str += "*";
	}
	str += "\n";
}
```

```
document.write(str);
```

Pozostaje poziom hard. Załóżmy że chcemy w konsoli wypisać:

```
******
*____*
*____*
******
```

Użyjmy do tego pętli w pętli, w której zbadamy dane liczniki:

```
let str = "";

for (let i=0; i<4; i++) {
        for (let j=0; j<6; j++) {
                if (i==0 || i==3 || j==0 || j==5) {
                        str += "*";
                } else {
                        str += "-";
                }
        }
        str += "\n";
}

document.write(str);
```

Break i continue

Czasami podczas pętli gdy wykonamy jakąś czynność - czy to wypiszemy dany element, czy go sprawdzimy, chcielibyśmy zakończyć dalsze wykonywanie takiej pętli. Służy do tego instrukcja **break**.

```
let str = '';
let i = 0;

while (i <= 100) {
        str += i;
        if (str.length > 20) break;
        i++;
}

document.write(str);
```

```
const tab = ["Ala", "Monika", "Beata", "Karol"];

let userExist = false;

for (let i=0; i<tab.length; i++) {
	if (tab[i] === "Beata") {
		userExist = true;
		break; //dalej nie ma sensu sprawdzać
	}
}

//to samo uzyskamy o wiele lepiej za pomocą includes(), indexOf() czy findIndex()


let nr = -1;

for (let i=0; i<1000; i++) {
	nr = Math.random();
	if (nr >= 0.95) break;
}

document.write(nr);
```

Drugą instrukcją jest **continue**. Nie przerywa ona działania pętli, a powoduje przerwanie danej iteracji (czyli aktualnego powtórzenia):

```
const tab = ["Ala", "Monika", "Beata", "Karol", "Alicja"];

for (let i=0; i<tab.length; i++) {
	if (tab[i] === "Karol") {
		continue; //Karola pomiń
	}
	document.write(tab[i]);
}


let i = 0;
let sum = 0;

while (i < 5) {
	i++;
	if (i === 3) continue;
	sum += i;
```

```
        document.write(i, `suma kolejnych liczb to ${sum}`);
    }
```

Zwróć uwagę, że gdy stosujemy **continue** w pętli while, zwiększanie licznika musimy robić przed użyciem tej instrukcji. Inaczej możemy trafić na moment, gdy aktualne powtórzenie będzie przerywane a tym samym zwiększanie licznika nigdy nie nastąpi.

```
    let i = 0;
    let sum = 0;

    while (i < 100) {
        i++;
        if (i % 2 === 0) continue; //gdy i jest parzyste przerywamy daną iteracją i przechodzimy do następnej
        sum += i;
    }
    document.write(i, `suma kolejnych liczb to ${sum}`);
```

Labele dla pętli

Każda pętla może być dodatkowo nazwana za pomocą etykiet. Dzięki nim możemy stosować instrukcje break i continue dla pętli o danej nazwie:

```
for (let i=0; i<10; i++) {
  for (let j=0; j<10; j++) {
    if (warunek) break; //normalnie mogę przerwać tylko pętlę w której użyłem instrukcji break/continue
  }
}
first:
for (let i=0; i<10; i++) {
  second:
  for (let j=0; j<10; j++) {
    if (warunek) break first; //przerywam główną pętlę
  }
}
loopA: for (let i=0; i<10; i++) {
  loopB: for (let j=0; j<10; j++) {
    if (warunek) continue loopA;
  }
}
```

Przy czym funkcjonalność ta jest tak skrajnie rzadko używana, że prawdopodobnie nigdy się z nią nie zetkniesz...

Edit: Tak naprawdę składnia ta zyskuje na popularności za sprawą https://svelte.dev/, który używa jej oznaczania reaktywnych deklaracji:

```
const a = 100;
const b = 200;

$: const nr = a * b; //label która ma nazwę "$"
```

Zadania

1. Wypisz w konsoli liczby od 0 do 9
2. Uzupełnij pętlę w taki sposób, aby kończyła swoje działanie w momencie, gdy wartość zmiennej i będzie równa 10

```
for (var i = 5; i < 16; i++) {
  console
  if () {

  }
}
```

3. Uzupełnij pętlę w taki sposób, aby w momencie, gdy wartość zmiennej i będzie równa 10 przeskoczy do kolejnego obiegu pętli, pomijając wypisanie tej wartości

```
for (var i = 5; i < 16; i++) {
  if () {

  }
  console
}
```

Interaktywnie sprawdź swoje rozwiązanie i znajdź więcej zadań na:

https://technikprogramista.pl/kurs/javascript/lekcja/javascript-petle-cwiczenia-interaktywne/