JavaScript:
cykle

dimon.work/kurs.html

1. Cykle - powtarzanie fragmentu kodu

Cykle w JavaScript

Cykle w językach programowania są sposobem na wielokrotne **powtarzanie fragmentu** (bloku) **kodu**. Liczba powtórzeń jest określana przez **warunek cyklu**; w zależności od warunku **cyklu** albo wykonuje jeszcze jedną **iterację** (powtarza fragment kodu w ciele cyklu), albo **kończy** swoją **pracę**.

2. Cykl do/while()

Jeśli niektóre działania muszą zostać powtórzone, ale nie wiadomo ile razy z góry

```
1
2       let pinCode;
3
4       do {
5
6           pinCode = prompt('Введите Код');
7
8           if(pinCode == '4321'){
9               alert('Open Door');
10          }else{
11              alert('Code Error');
12          }
13
14      } while(pinCode != '4321');
15
```

Jeśli **kod** nie pasuje, musimy ponownie zażądać go od użytkownika i tak powtarzać, aż zostanie wprowadzony prawidłowy kod. Tzn. potrzebujemy mechanizmu, który będzie **powtarzał zestaw działań, dopóki warunek nie będzie prawdziwy**
(na przykład: hasło nie jest równe«4321»).

Cykl **do/while()** umożliwia powtarzanie kodu według warunku. Warunek jest sprawdzany na końcu każdej **iteracji** (kroku) pętli i określa, czy pętla przejdzie do kolejnej „rundy”.

3. Cykl while()

Cykle - sposób na wielokrotne powtarzanie fragmentu kodu

```
1
2       let currentTemperature  = 25;
3       let temperatureLimit    = 100;
4
5
6       while(currentTemperature < temperatureLimit){
7           console.log(`Temperature ${currentTemperature} °C`);
8
9           currentTemperature++;
10      }
11
12      console.log(`Heating ended at ${currentTemperature} °C`);
13
14
```

Cykl **while()** umożliwia powtarzanie kodu według warunku. Warunek jest sprawdzany przed rozpoczęciem każdej **iteracji** (kroku) cyklu i określa, czy cykl będzie kontynuowała tę „rundę”. Cykl **while** (podobnie jak **do/while**) wykonuje fragment kodu tak długo, jak warunek w niej określony jest **true**.

Główną różnicą między **while** i **do/while** jest to, że ten pierwszy jest odpowiedni dla zadań, w których nie może zostać wykonany ani jeden „przebieg” pętli, podczas gdy ten drugi jest odpowiedni dla zadań, w których musi zostać wykonany co najmniej jeden **krok** (**iteracja**) pętli.

Kluczowym punktem stosowania cykli w JavaScript

Ciało cyklu musi zawierać zmiany zmiennych, które są w warunku, w przeciwnym razie pętla będzie działać w nieskończoność

4. Cykl for()

Cykl for – kiedy wiadomo ile raz chcemy powtórzyć kod

```
1
2       let steps = 10;
3
4       for(let i = 1; i <= steps; i++){
5           console.log(`Step ${i} of ${steps}`);
6       }
7
```

Cykl **for** jest wygodna w przypadkach, gdy z góry wiadomo (lub można obliczyć na podstawie już dostępnych danych), ile razy należy powtórzyć określoną czynność.
Cykl **for** jest również nazywana „pętlą licznikową” - nazwa jest warunkowa.

5. Operatory continue/break

Operatory break/continue

```
1
2       let steps = 100;
3
4       for(let i = 1; i <= steps; i++){
5
6
7           if( i % 3 == 0 ) {
8               continue;
9           }
10
11          if( i == 17 ) {
12              break;
13          }
14
15          console.log(`Step ${i} of ${steps}`);
16
17      }
18
```

Operator **break** powoduje wcześniejsze zakończenie cyklu, bezpośrednio w punkcie wywołania. Nie ma to sensu bez towarzyszącej instrukcji **if/else**.

Operator **continue** zmusza pętlę do zakończenia bieżącej iteracji i natychmiastowego przejścia do następnej. Nie ma on również sensu bez towarzyszącego mu operatora **if/else**

6. Trochę praktyki #1

Gra „Zgadnij liczbę”

Skrypt **losuje liczbę** (od 1 do 100 włącznie) i daje graczowi **10 prób** odgadnięcia jej, jeśli użytkownik nie zgadnie - skrypt informuje, że przegrał, a gra mówi mu, jaka była prawidłowa odpowiedź. Jeśli gracz zgadł, gra informuje, że wygrał. Do generowania liczb użyj funkcji **Math.random();** Gra powinna podpowiadać, czy wprowadzona przez użytkownika liczba jest większa lub mniejsza od odgadniętej.

7. Trochę praktyki #2

Depozyt z kapitalizacją

Dostępne są dane (wprowadzone przez użytkownika) dotyczące **kwoty lokaty**, **stopy procentowej** (rocznej) i **okresu lokaty w miesiącach**. Konieczne jest obliczenie, ile **dochodu** przyniesie lokata z miesięczną kapitalizacją odsetek na koniec okresu.

Weź pod uwagę, że odsetki są naliczane co miesiąc, liczba dni w miesiącu i w roku nie wpływa na wynik, podatki również nie są brane pod uwagę.

Zadanie domowe

Zadanie #B1

Месяц	Задолженность по кредиту	Погашение кредита	Проценты по кредиту	Комиссии	Выплаты в месяц
1	1000.00	83.40	20.90	0.00	104.30
2	916.60	83.40	19.10	0.00	102.50
3	833.20	83.40	17.40	0.00	100.80
4	749.80	83.40	15.70	0.00	99.10
5	666.40	83.40	13.90	0.00	97.30
6	583.00	83.40	12.20	0.00	95.60
7	499.60	83.40	10.50	0.00	93.90
8	416.20	83.40	8.70	0.00	92.10
9	332.80	83.40	7.00	0.00	90.40
10	249.40	83.40	5.20	0.00	88.60
11	166.00	83.40	3.50	0.00	86.90
12	82.60	82.60	1.80	0.00	84.40
Итого		1000.00	135.90	0.00	1135.90

Wykonanie obliczeń schematu spłaty pożyczki zgodnie z klasycznym schematem. Parametry określające kredyt (dane wejściowe): **kwota**, **stopa procentowa** (% rocznie) i **okres kredytowania** (w miesiącach). Prowizje i inne opłaty nie są brane pod uwagę.

Program powinien zawierać informacje dla każdego miesiąca: jaka część pożyczki powinna zostać spłacona, ile odsetek powinno zostać zapłaconych (oraz kwota części + odsetki) i ile pozostało na części pożyczki po tej spłacie.

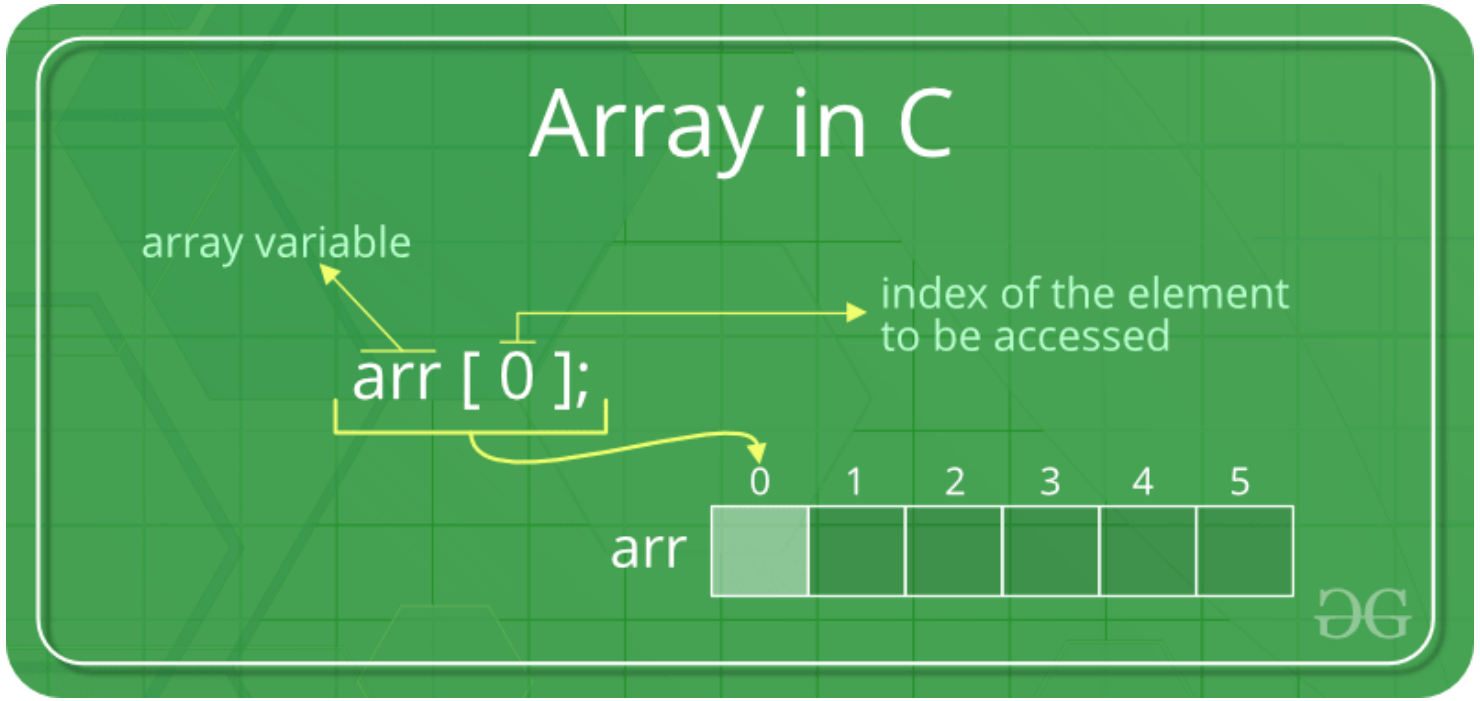
Na koniec należy wydedukować, jaka będzie **nadpłata** pożyczki.

Wszystkie kwoty pieniężne muszą być zaokrąglone do 2 miejsc po przecinku.

Przykład funcjonału:
https://fin-calc.org.ua/en/credit/calculate/

На следующем занятии

JS: циклы и массивы



Хранения и обработка наборов данных