Interpreter – TUTORIAL

W odróżnieniu do normalnego JS, deklaracje i definicje zmiennych, obiektów, tablic muszą zostać zakończone średnikiem ‘;’

1. Deklaracja zmiennych.

Zmienne można deklarować przy pomocy słowa kluczowego: var, np.

Dobrze:	**Źle:**
var x;	x;
var x, y;	x, y;

Deklaracja pojedyńczej zmiennej może odbyć się bez użycia var w trakcie przypisania, tzn.

Dobrze:

x = 12;

var x = 12;

- stworzy zmienną x z przypisaną wartością

Nie mozna deklarować wielu zmiennych, nawet gdy przypisujemy wartość, bez użycia var:

Dobrze:	**Źle:**
var x, y = 12;	x, y = 12;

Deklaracja zmiennej bez przypisania jej wartości, w przypadku istnienia zdefiniowanej o tej samej nazwie, zwróci istniejącą, tzn:

var x = 5;	lub	x = 5;
var x;		var x;

- zmienna x nadal będzie miała wartość 5

2. Instrukcje przypisania.

Zmienne, tak jak w JS są dynamiczne. Można przypisywać im integery, float, string oraz boolean.

Dobrze:

x = 5;

x = “zastap 5”;

x = true;

x = 5.4;

Zmiennym można przypisać dowolne inne zmienne:

Dobrze:

x = 15;

y = x;

Zmiennym można przypisać: obiekty, listy, funkcje, wyniki wyrażeń logicznych lub działań.

3. Operatory i obsługiwane działania.

Obsługiwane są operacje z wykorzystaniem operatorów:

‘+’	- dodawanie	‘==’	- równy
‘-’	- odejmowanie	‘!=’	- różny
‘*’	- mnożenie	‘&=’	- równy z uwzględnieniem typu
‘/’	- dzielenie	‘>’	- większy
‘!’	- zaprzeczenie	‘<’	- mniejszy

‘-()’ - przeciwna wartość ‘>=’ - wiekszy/równy
‘++’ - pre-inkrementacja ‘<=’ - mniejszy/równy
‘--’ - pre-dekrementacja ‘=’ - przypisanie
Operatory: **‘-’, ‘*’, ‘/’, ‘++, ‘--’, ‘-()’** działają tylko ze zmiennymi z przypisanymi wartosciami **int** lub **float**.

Operator **‘!’** działa tylko ze zmiennymi typu **boolean.**

Operator **‘+’**, można użyć przy zmiennych:
- (int) + (int)
- (float) + (float)
- (int) + (float)
- (bool) + (bool) – true przyjmie wartość 1, a false 0, np. true + true = 2
- (string) + (string) - konkatenacja
- (int) + (string) – konkatenacja, int zamieniany jest na stringa

Operatory **‘==’** oraz **‘!=’** działają z:
- (int) == (int)
- (float) == (float)
- (int) == (float)
- (bool) == (bool)
- (int) == (bool) – dla true = 1 oraz false = 0, np. 1 == true zwróci true
- (string) == (string)
- (int) == (string) – np. 1 == “1” zwróci true

Operator **‘&=’** jest odpowiednikiem JS-go ‘===’, czyli sprawdzania także typu, np.
1 &= “1” - zwróci tym razem false
1 &= true – zwróci false
“1” &= “1” - zwróci true, itd.

Wyniki operatorów mogą zostać przypisane zmiennym:
x = 4 < 5; // x == true
x = 6 <= 5; // x == false
x = “e” <= “d”; // x == false
x = “c” > “b”; // x == true
x = “5” == 5; // x == true
x = true > false; // x == true
x = false <= false; // x == true

4. Obiekty.
Deklarowane przy pomocy {}, np.
x = {}; - pusty obiekt

Można zdefiniować klucz : wartość, oddzielone ‘,’:
x = {a: 12, b: 1};

Kluczem może być dowolny wyraz, zaczynający się na literę a-zA-Z. Wartością mogą być typy standardowe, obiekty, listy, funkcje, np.
```
x = {a: {
        b: “wew”
    },
```

```
    b: [1, 3],
    c: function (arg){ return arg * 2;}
};
```

Wartosć zwrócona przez [nazwa_obiektu].[klucz], np. Dla powyższego obiektu:
y = x.a.b; // y == “wew”
y = x.b[0]; // y == 1

5. Listy.

Deklarowane przy pomocy [], np.
x = []; - pusta lista

Elementami list mogą być jak w przypadku obiektów: podstawowe typy, inne listy, obiekty, funkcje anonimowe, oddzielone ‘,’, np.
y = {a: 123, b: [“a”, “b”]};
x = [[1, “elem”], y, “45”, 54, function(arg){return arg / 2;}];

Dostęp do wartości uzyskujemy standardowo przez **[pozycja_w_liscie]**, gdzie podana pozycja może być tylko liczbą. Jeśli podamy pozycję wykraczającą poza rozmiar listy, zwrócone zostanie 0. Pozycje numerowane są od **0**, np. dla powyższej listy:
z = x[0][1]; // z == “elem”
z = x[1].b[0]; // z == “a”
z = x[4](10); // z == 5
z = x[10]; // z == 0

Dodatkowo istnieją funkcje **push()** - dodaje element na końcu listy, oraz **size()** - zwraca wielkość listy, wywoływane na liście przy pomocy ‘.’, np:
x = [];
z = x.size(); // z == 0
x.push(1);   // x == [1]
z = x.size(); // z == 1

6. Funkcje.

Tworzone za pomocą słowa kluczowego **function,** np.
function sum(a, b) { return a + b;}

Używane ze słowem kluczowym **return,** zwracającego wartość.
x = sum(2, 5); // x == 7

Mogą być zdefiniowane anonimowo:
x = function (arg) {return arg + 5;};
y = x(1); // y == 6

Wewnątrz ciała funkcji można deklarować nowe zmienne. Wszystkie zmienne są lokalne w ciele funkcji, tzn. niewidoczne są zmienne o tej samej nazwie z zewnątrz, np.
x = 5;
y = function (arg) {x = 2; return arg + x;};
z = y(2); // z == 4, x == 5

Poza deklarowaniem zmienych i zwracaniem wartości, wewnątrz funkcji można wykonywać wszystkie inne operacje.

Dostępna jest globalna funkcja **log()**, przyjmująca wyłącznie jeden parametr, który wyswietla na ekranie użytkownika, np.
log(5); - wyświetli 5 na ekranie
x = [9, 5, “d”];
log(x) – wyświetli [9 , 5 , “d”]

Nie można użyć funkcji, gdy podamy więcej parametrów, np. log(“zawartosc x: ”, x), zwróci błąd

7. Pozostałe konstrukcje językowe.

Pętle.
Obsługiwane są pętle while w postaci:

while(‘warunek’) {‘ instrukcje do wykonania’}, gdzie warunek to dowolny warunek logiczny, np.

```
x = 0;
while(x < 5){
  log(x);
  ++x;
}
```

Instrukcje warunkowe.
Tworzone za pomocą słowa kluczowego **if** oraz **else.**
Można stworzyć na dwa sposoby:

if(‘warunek’){‘ instrukcje do wykonania’}

lub

if(‘warunek’) {‘instrukcje do wykonania jesli warunek poprawny’}
else{‘ instrukcje do wykonania jeśli niepoprawny’}

np.

```
x = 3;
if(x < 5) {
   log(“x < 5”);
} else {
  log(“x >= 5”);
}
```