Funkcje

Wydzielanie części kodu

W programowaniu wielokrotnie wykonuje się te same lub podobne operacje. Naturalnym jest, że zamiast pisać je po wielokroć, wolelibyśmy napisać powtarzalny kod tylko raz, a potem go wywoływać kiedy tylko chcemy. Właśnie tutaj z pomocą przychodzą funkcje.

Żeby je lepiej zrozumieć wyobraź sobie Jacka, który bardzo lubi hamburgery. A więc na śniadanie, obiad i kolację je hamburgera. Aby zjeść takiego hamburgera wymagane jest jednak kilka kroków przygotowań: podsmażenie mięsa, podsmażenie bułki, dodanie pomidora i wreszcie sama konsumpcja. Jego dzień można opisać następująco:

```
console.log("Wstań z łóżka");
console.log("Umyj zęby");
console.log("Podsmaż mięso");
console.log("Podsmaż bułkę");
console.log("Dodaj pomidora");
console.log("Zjedz hamburgera");
console.log("Pracuj");
console.log("Podsmaż mięso");
console.log("Podsmaż bułkę");
console.log("Dodaj pomidora");
console.log("Zjedz hamburgera");
console.log("Pracuj");
console.log("Podsmaż mięso");
console.log("Podsmaż bułkę");
console.log("Dodaj pomidora");
console.log("Zjedz hamburgera");
console.log("Graj w gry");
console.log("Idź spać");
```

Powtarzalny proces przygotowywania i konsumpcji hamburgera nie tylko jest uciążliwy, ale także zaciemnia obraz tego, jak naprawdę wygląda dzień Jacka.

Możemy poprawić ten kod poprzez wyciągnięcie funkcji makeAndEatHamburger. Jej ciało będzie zawierało poszczególne kroki związane z przygotowaniem i konsumpcją, dzięki czemu będziemy mogli je uruchomić poprzez wywołanie tej funkcji.

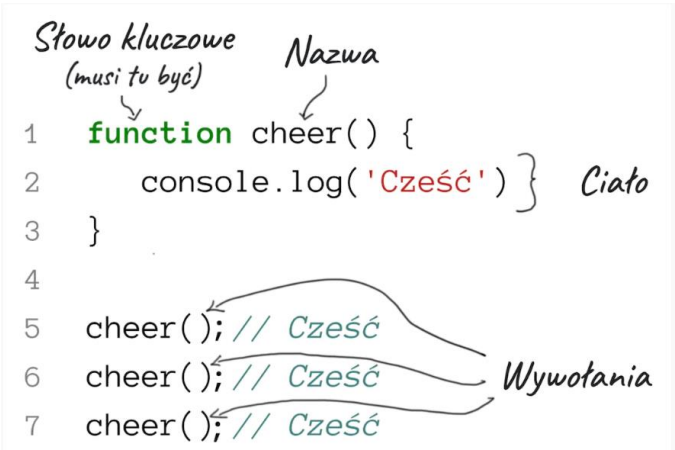
```
// Definicja funkcji
function makeAndEatHamburger() {
  console.log("Podsmaż mięso");
  console.log("Podsmaż bułkę");
  console.log("Dodaj pomidora");
  console.log("Zjedz hamburgera");
}

console.log("Wstań z łóżka");
console.log("Umyj zęby");
makeAndEatHamburger(); // Użycie funkcji
console.log("Pracuj");
makeAndEatHamburger(); // Użycie funkcji
console.log("Pracuj");
makeAndEatHamburger(); // Użycie funkcji
console.log("Graj w gry");
console.log("Idź spać");
```

Jak pisać?

Standardowa funkcja zaczyna się od słówka **function**, potem następuje jej **nazwa**, **nawiasy zwykłe**, w których **mogą** znajdować się **definicje parametrów**, i wreszcie **ciało funkcji**, które umieszczamy w nawiasach klamrowych.
Ciało zawiera zbiór instrukcji, które zostaną kolejno wywołane, gdy wywołana zostanie funkcja.
Następnie **wywołujemy zdefiniowaną funkcję przez użycie jej nazwy, a po niej nawiasów okrągłych**.

Zwróć uwagę na różnicę między definicją funkcji a jej wywołaniem. Definicja zaczyna się od słówka function i określa, co pod daną nazwą powinno stać. Wywołanie to po prostu nazwa funkcji i nawias okrągły. Ono wywołuje kod zawarty w definicji. W ciele funkcji możemy użyć wszystko, czego nauczyliśmy się do tej pory — zmienne, pętle czy nawet same funkcje.



Funkcja jako wyznacznik wielu operacji

Użycie funkcji do ukrycia wielu operacji pod pojedynczą nazwą jest bardzo ważne. Pomyśl tylko o dowolnym działaniu, na przykład wyjściu do sklepu. Co robisz? Wkładasz buty, ubierasz się w kurtkę, wychodzisz z domu... ale każda z tych operacji to składowa wielu mniejszych. Czym jest na przykład zakładanie butów? Nałożenie na nogę, wiązanie sznurówek... Czym jest to wiązanie sznurówek? Jest to skoordynowana sekwencja ruchów wielu mięśni. Myślimy o abstrakcyjnych działaniach, pod którymi stoi wiele mniejszych działań.

W programowaniu to funkcje służą do ukrywania sekwencji działań za pojedynczą nazwą.

```
function goToStore() {
  wearShoes()
  putOnJacket()
  leaveHouse()
  // ...
}

function wearShoes() {
  putOnShoe()
  tieShoelace()
  // ...
}

// ...
```

Funkcja może reprezentować nawet bardzo złożone czynności, a sama być prosta, bo używa tylko nieco mniej złożonych elementów. Tak właśnie skonstruowane są programy.

Jak działa funkcja?

Przeanalizujmy krok po kroku jak działają funkcje. Program wykonuje kolejne instrukcje linijka po linijce. Gdy wywoływana jest funkcja, przeskakujemy na sam początek jej ciała i od tego miejsca idziemy dalej. Gdy dojdziemy do końca (albo słówka return, o którym powiemy niedługo), wracamy do miejsca, gdzie funkcja była wywołana i stamtąd kontynuujemy. Tak w skrócie działają funkcje. Poniżej znajduje się prezentacja przykładowych funkcji wraz z kolejnymi numerami reprezentującymi kolejność ich wywołania.

```
function firstFunction() {
  console.log("3"); // 3
  console.log("4"); // 4
}

console.log("1"); // 1
console.log("2"); // 2
firstFunction();
console.log("5"); // 5
secondFunction();
console.log("8"); // 8

function secondFunction() {
  console.log("6"); // 6
  console.log("7"); // 7
}

// Wypisze się kolejno 1 2 3 4 5 6 7 8
```

Warto spędzić chwilę i poćwiczyć wyobrażanie sobie jak idzie nasz program po kolejnych linijkach, jak przeskakuje do funkcji, a potem jak przeskakuje z powrotem do miejsca wywołania.

Odnoszenie się do elementów spoza funkcji

W funkcji możemy korzystać ze wszystkiego, co zostało zdefiniowane — na przykład ze zmiennej albo innych funkcji.

```
let whoToCheer = "Czytelniku";

function cheer() {
  console.log("Cześć " + whoToCheer + "!");
}

cheer(); // Cześć Czytelniku!
whoToCheer = "Wszystkim";
cheer(); // Cześć Wszystkim!

function cheerTwoTimes() {
  cheer();
  cheer();
}

cheerTwoTimes(); // Wypisze:
// Cześć Wszystkim!
// Cześć Wszystkim!
```

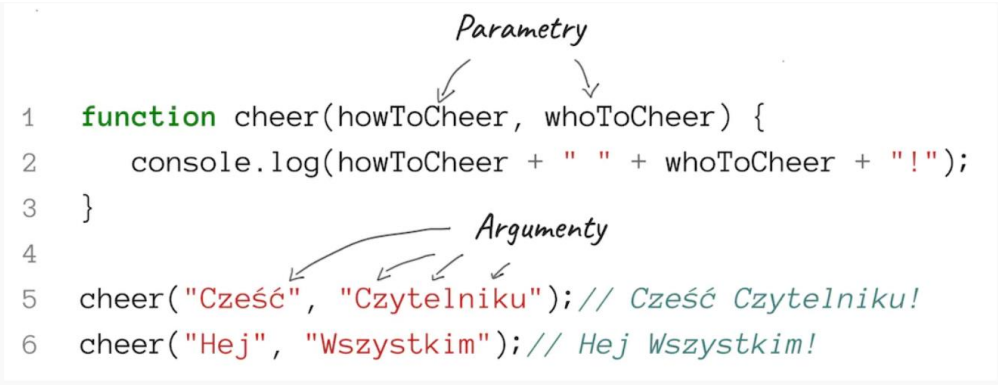
Parametry i argumenty funkcji

Większość funkcji chciałaby mieć jednak wartości tylko dla siebie. Jest to możliwe: wystarczy zdefiniować zmienne wewnątrz nawiasów w definicji funkcji. **Takie zmienne nazywane są parametrami**. Nie wymagają one żadnego słówka let i definiują to, co powinno być przekazane do konkretnej funkcji. Z drugiej strony, przy wywołaniu funkcji powinniśmy podać wartości, które będą przekazane do tych parametrów. Takie wartości nazywane są argumentami. W poniższym przykładzie whoToCheer to parametr funkcji, a "Czytelniku", "Wszystkim" oraz 42 to wartości używane jako argumenty.

```
function cheer(whoToCheer) {
  console.log("Cześć " + whoToCheer + "!");
}

cheer("Czytelniku"); // Cześć Czytelniku!
cheer("Wszystkim"); // Cześć Wszystkim!
cheer(42); // Cześć 42!
```

Funkcja może mieć więcej parametrów, a wywołanie składać się z większej liczby argumentów — w takim przypadku oddzielamy je przecinkami.



Brakujące lub nadmiarowe argumenty

JavaScript pozwala, abyśmy wywołali funkcję z mniejszą liczbą argumentów, niż funkcja się spodziewa. Parametry, dla których zabrakło argumentów, przyjmą wtedy wartość undefined, podobnie do zwykłych zmiennych, których wartości nie zostały określone. Nadmiarowe argumenty zostaną natomiast zignorowane.

```
function printAll(a, b) {
  console.log(a + ", " + b);
}

printAll("A", "B", "C"); // A, B
printAll("A", "B"); // A, B
printAll("A"); // A, undefined
printAll(); // undefined, undefined
```

Wynik funkcji – return

Poznaliśmy funkcje już na lekcjach matematyki. Tam były one definiowane, aby obliczyć wynik dla wartości wejściowych. W szkole poznaliśmy funkcje podnoszące do kwadratu, modulo czy silni. Możemy takie funkcje zdefiniować również w programowaniu. Aby zwrócić wartość z funkcji, musimy użyć słowa **return** i dzięki temu zostanie ona zwrócona z wywołania tej funkcji.

```
function returnNumber() {
  return 42;
}

const result = returnNumber();
console.log(result); // 42
```

return natychmiast kończy wywołanie funkcji i zwraca wynik.

Zadanie 1

a) Zdefiniuj funkcję do obliczenia kwadratu liczby (ang. square).

Odp.:

```
function square(x) {
  return x * x;
}
```

b) Zdefiniuj funkcję zwracającą wartość pola prostokąta

Zadanie 2

a) Napisz funkcję zwracającą wartość bezwzględną. Ta funkcja powinna zwracać:

- wartość wejściową, gdy jest ona większa lub równa zero,
- przeciwność wartości wejściowej, gdy jest ona mniejsza od zera.

Odp.:

```
function absolute(x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  }
}
```

b) Napisz funkcję zwracającą wartość silni. Dla przykładu silnia z 5 jest równa 1 * 2 * 3 * 4 * 5, a więc 120.

Funkcje matematyczne

Pisanie funkcji matematycznych jest dobrym ćwiczeniem, ale by każdy programista nie powtarzał tej samej pracy, te najważniejsze zostały już zdefiniowane w samym JavaScript. Aby je wywołać, powinniśmy zacząć od Math, a potem użyć skróconej nazwy funkcji. Oto kilka przydatnych funkcji:

- *Math.abs*(x) - wartość bezwzględna,
- *Math.pow*(x, y) - potęga (jest to więc alternatywa do operatora **),
- *Math.min*(x, y), Math.min(x, y, z, ...) - najmniejsza z wartości,
- *Math.max*(x, y), Math.max(x, y, z, ...) - największa z wartości,
- *Math.log*(x), *Math.log2*(x), *Math.log10*(x) - odpowiednio: logarytm naturalny, logarytm o podstawie 2, logarytm dziesiętny.
- *Math.sin*(x), *Math.cos(*x), ... - sinus, cosinus itp.

Obiekt ten zawiera również kilka stałych, na przykład **Math.PI**.

Dodatkowo bardzo przydatną funkcją jest Math.random(), która generuje losową wartość między 0 a 1 (bez 1). Bardzo przydaje się, gdy chcemy do programu wprowadzić pewną losowość.

Funkcje jako wartości

Kiedy definiujemy funkcję o danej nazwie, tak naprawdę definiujemy zmienną o takiej nazwie. Zmienną, którą można wypisać lub też przypisać do innej zmiennej.

```
function add(a, b) {
  return a + b;
}

console.log(add(1, 2)); // 3

console.log(add);
// Wypisze:
// ƒ add(a, b) {
//   return a + b
// }

console.log(typeof add); // function

const plus = add;
console.log(plus(1, 2)); // 3
```

Zauważ, że kiedy wywoływaliśmy funkcję, zarówno teraz, jak i wcześniej, to tak naprawdę wywoływaliśmy zmienną. Jeśli zmienna reprezentuje funkcję, to można za nią postawić nawias okrągły, by ją wywołać. Tak więc nic nie stoi na przeszkodzie, by wywołać plus(1, 2).

Kiedy definiujemy funkcję, nie musimy nadawać jej nazwy. Definicja funkcji sama zwraca odwołanie do tej funkcji jako wartości, a to możemy przypisać do zmiennej.

```
const add = function(a, b) {
  return a + b;
}
console.log(add(1, 2)); // 3
```

Funkcje, które nie zawierają w swojej definicji nazwy, znane są jako funkcje anonimowe, czyli bezimienne (jak nasz Gall Anonim). Można by argumentować, że przecież funkcja w powyższym przykładzie ma nazwę add. Czasem dla uproszczenia tak się mówi, ale będąc bardziej precyzyjnym, funkcja ta nie ma nazwy (jest anonimowa), za to wskazuje na nią zmienna o nazwie add.

```
function add1(a, b) { // funkcja nazwana
  return a + b;
}
console.log(add1(1, 2)); // 3

const add2 = function(a, b) { // funkcja anonimowa
  return a + b;
}
console.log(add2(1, 2)); // 3
```