HTML
CSS
JS

WSTĘP TEORETYCZNY

INFORMACJE WSTĘPNE

FRONT-END DEVELOPER

Programista odpowiedzialny za **kod determinujący końcowy wygląd strony czy aplikacji** webowej. Innymi słowy, jest odpowiedzialny za **obsługę interfejsu projektu**, dopasowanie go do różnego rodzaju urządzeń (RWD) oraz dodanie i animację elementów graficznych.

BACK-END DEVELOPER

Programista odpowiedzialny za niewidoczną na pierwszy rzut oka część aplikacji. Odpowiada za **funkcjonalność techniczną projektu**, to znaczy za poprawne działanie systemu, nadzór nad jego bezpieczeństwem i rozwojem oraz za zaplecze serwerowe aplikacji czy strony.

JAK WYGLĄDA WSPÓŁPRACA GRAFIKA I FRONT-END DEVELOPERA?

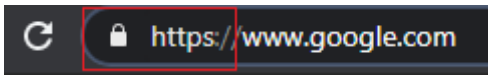
Grafik tworzy wygląd strony, jej układ (*layout*), pracuje nad oprawą wizualną. **Developer przekłada projekt grafika na kod** tak, żeby serwer i przeglądarka mogły go zinterpretować. Do języków wykorzystywanych we front-endzie należą HTML, CSS, JavaScript.

HTTP

Hypertext Transfer Protocol. Protokół dotyczący **przesyłu danych (zasobów) i zasad komunikacji pomiędzy klientem, a serwerem**. Protokół HTTP określa w jaki sposób informacje są udostępniane, przetwarzane i odczytywane przez serwer oraz jak wygląda odpowiedź na żądanie.

UWAGA! W przypadku stron, które gromadzą nasze poufne dane (w tym dane logowania) warto zwrócić uwagę na to, czy adres strony rozpoczyna się od http czy https. HTTPS to szyfrowana wersja protokołu HTTP (przy pomocy protokołu SSL/TLS). Dzięki temu **poufne dane są zabezpieczone i trudniejsze do przechwycenia**. Szczególnie istotne

w przypadku stron bankowych, wykonywania płatności online czy stron obsługujących dane wrażliwe. O zastosowaniu protokołu HTTPS może również świadczyć ikona kłódki na pasku zadań.



CZYLI W JAKI SPOSÓB PRZEGLĄDARKA WYŚWIETLA ZAWARTOŚĆ STRONY?

W pewnym uproszczeniu - w momencie, w którym wpisujemy w pasku przeglądarki adres strony przeglądarka lokalizuje adres IP dla danej domeny. Wysyła zapytanie do odpowiedniego serwera, który odnajduje dokument .html. Komplet danych jest odsyłany do przeglądarki, która interpretuje kod i wyświetla go użytkownikowi w przyjaznej formie.

KODOWANIE A PROGRAMOWANIE

KODOWANIE to **tłumaczenie języka ludzkiego na język maszynowy.** Innymi słowy – zapisywanie odpowiedniego kodu, który zostanie zinterpretowany przez komputer. Dobrym przykładem są tutaj HTML i CSS, które pozwalają na łatwe przełożenie projektu grafika na kod, aby przeglądarka mogła go odpowiednio wyświetlić. W tym wypadku wystarczające są podstawowe narzędzia – taki kod można zapisać w notatniku i wciąż uzyskamy efekt końcowy.

PROGRAMOWANIE to **proces, podczas którego tworzony jest wykonywalny program**. To nie tylko zapisywanie kodu, ale również planowanie aplikacji, samo projektowanie, testowanie, wdrażanie i utrzymanie aplikacji. **Program powinien wykonywać określone zadanie i spełniać założone standardy.** W przypadku programowania najczęściej stosowane są dodatkowe narzędzia, jak np. kompilatory, debuggery itd. Programista powinien znać założenia i składnię języka i wykorzystywać go w taki sposób, aby uzyskać działający produkt.

FRAMEWORK/BIBLIOTEKA

„Programista wywołuje bibliotekę, a framework wywołuje programistę."

Dobrze obrazuje to poniższa grafika[1]:

BIBLIOTEKA pozwala wywołać daną funkcję w momencie, gdy jest ona potrzebna. Jest to zbiór takich funkcji (i nie tylko) i mogą być wywołane w dowolnym momencie, w którym będą potrzebne.

FRAMEWORK to z kolei **schemat programu lub jego części.** Kod został już napisany, a teraz jest wykorzystywany i dostosowany do potrzeb (np. poprzez dodanie funkcjonalności lub pewną modyfikację już istniejących).

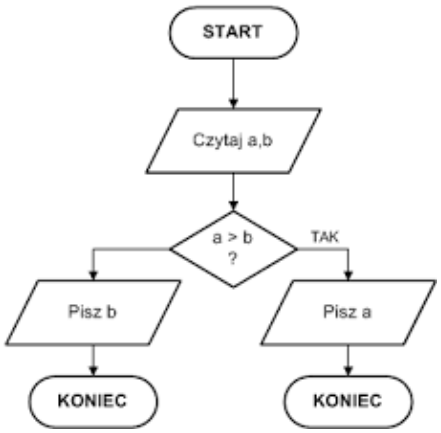
UWAGA! W języku Polskim oba te pojęcia często są stosowane zamiennie, nie zawsze zgodnie ze stanem faktycznym.

ALGORYTM

Algorytm to po prostu **uporządkowany ciąg jasno zdefiniowanych czynności, które prowadzą do wykonania określone zadania lub osiągnięcia celu.** Opisuje proces

[1] https://kamil.kwapisz.pl/wp-content/uploads/2020/04/framework_vs_biblioteka.png

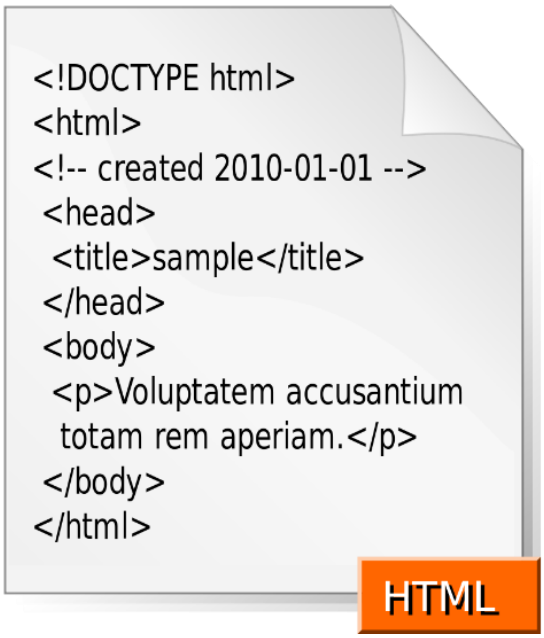
postępowania. Dobrą analogią jest przepis kucharski – krok po kroku wykonujemy kolejne czynności np. w celu upieczenia ciasta. Algorytm pozwala nam zorientować się w tym, jakie elementy powinny zostać poruszone podczas programowania tak, aby program działał w sposób jakiego oczekujemy.



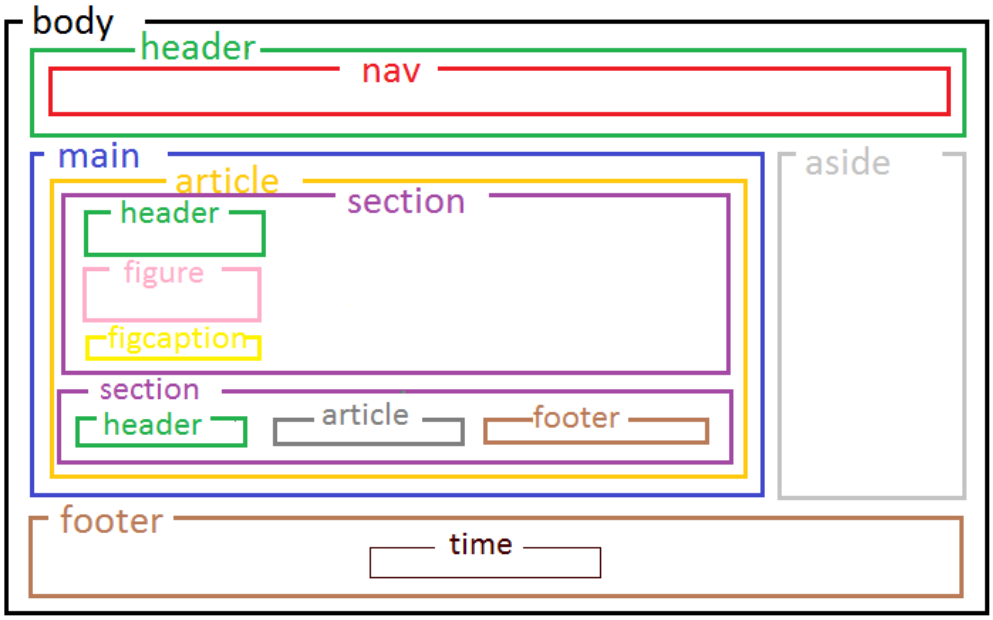
HTML

HyperText Markup Language, czyli hipertekstowy język znaczników. **Dzięki niemu możemy stworzyć strukturę strony internetowej czy aplikacji. Odpowiednie fragmenty okolone są przez odpowiednie znaczniki (tagi)** – w ten sposób powstają kolejne hiperłącza, nagłówki, listy, akapity, formularze i inne.

Kod zapisany za pomocą HTML powinien mieć odpowiednią budowę. Tylko poprawny zapis gwarantuje, że przeglądarka bez problemów zinterpretuje zakodowaną zawartość.



W HTML5 stosuje się znaczniki, które w nazwie sugerują już czego dotyczy kod. Np. <section></section> w przypadku kolejnych sekcji, <header></header> w przypadku nagłówków itd. Znaczniki pozwalają na stworzenie odpowiedniej architektury strony i wskazują przeglądarce czego dotyczy kod. Niżej przykład budowy strony i zastosowania znaczników.



CSS

Cascading Style Sheets czyli kaskadowe arkusze stylów. Za pomocą kodu zapisanego w CSS możemy wizualnie modyfikować stronę. **Ten język pozwala na stosowanie konkretnego stylu (modyfikacji) do określonych przez nas elementów w dokumentach HTML.** W ten sposób możemy np. zmienić kolor wszystkich nagłówków na stronie.

Dlaczego **kaskadowe arkusze stylów**? Ponieważ wygląd strony może być wynikiem kilku arkuszy stylów. Mamy do czynienia z arkuszami zewnętrznymi, możemy również mieć style wprowadzone w kod HTML (do danego znacznika można dodać atrybut style=”…”), poza tym każda przeglądarka ma wbudowane własne arkusze stylów. **Możemy nadpisywać style, a przeglądarka wybiera ten, który jest najważniejszy** (np. ten wpisany w kod HTML będzie miał wyższą wagę niż ten w pliku .css).

Podsumowując. **HTML odpowiada za architekturę i układ strony, tzw. layout. CSS z kolei odpowiada za ostateczny wygląd strony pod względem estetycznym.**

JAVASCRIPT

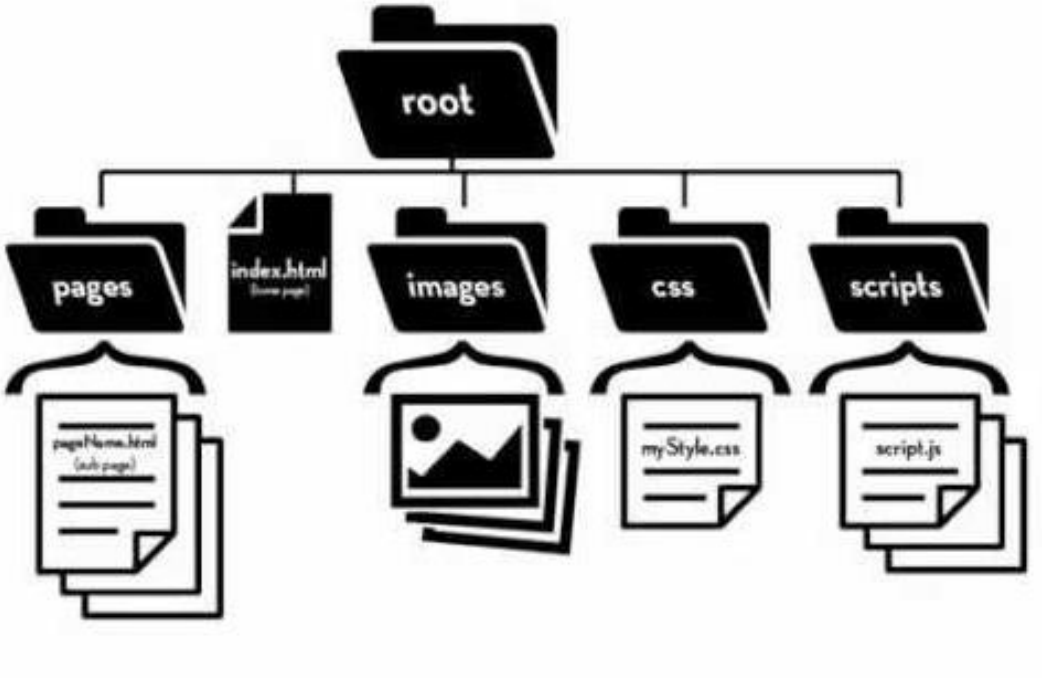
Skryptowy język programowania. **Pozwala na wykonywanie „akcji” na stronie** – jego najważniejszymi elementami są funkcje. Skrypty w tym języku umożliwiają m.in.: **reagowanie na zdarzenia** (np. najechanie kursorem myszy czy kliknięcie), **walidację danych wprowadzanych w formularzach** (np. komunikat, że wpisany ciąg znaków nie jest adresem mailowym) **oraz tworzenie złożonych efektów wizualnych**. JavaScript pozwala również na tworzenie pełnoprawnych aplikacji.

STRUKTURA PLIKÓW STRONY WWW

Tworząc stronę należy zachować strukturę plików i folderów. Główny plik HTML powinien **każdorazowo przyjmować nazwę index.html** – w ten sposób przeglądarka wie, w którym miejscu powinna rozpocząć interpretację nadesłanych przez serwer danych.

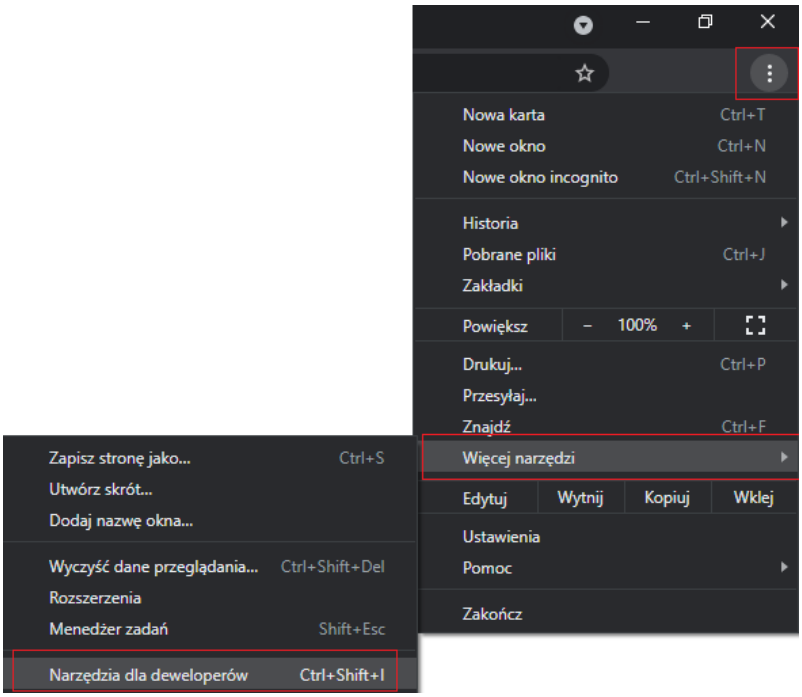
Najważniejsze foldery i nazwy plików:

- css/style.css,
- scripts/script.js,
- images/image.jpg,
- pages/dowolna_nazwa.html (tylko główny plik .html musi nazywać się index.html).

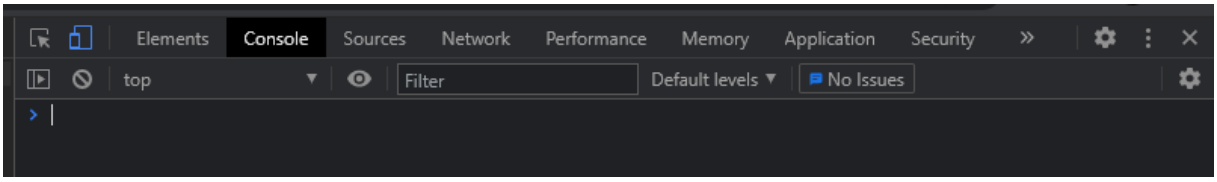


WSTĘP DO JAVASCRIPT – KONSOLA W GOOGLE CHROME

Aby otworzyć konsolę w Google Chrome należy wykorzystać skrót **Ctrl+Shift+i** lub znaleźć w przeglądarce **narzędzia dla developerów.**



Pojawia się okno z narzędziami developerskimi, w którym pasek zadań wygląda jak na rysunku poniżej.

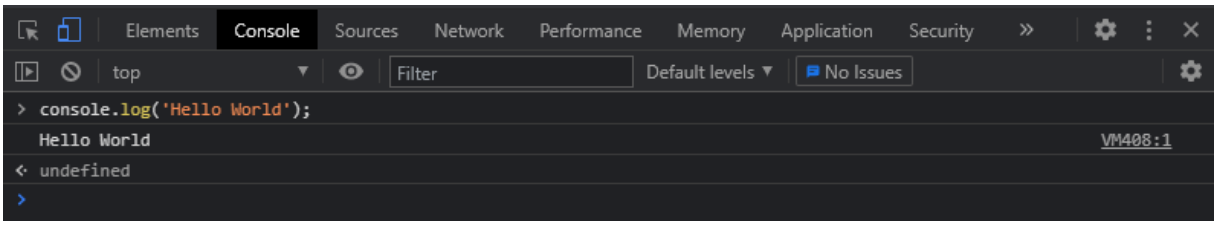


Aby spojrzeć na **całościowy kod strony** (HTML i CSS) warto zwrócić uwagę na zakładkę **Elements**, **strzałka w lewym górnym rogu pozwala na wskazywanie kolejnych elementów bezpośrednio na stronie**, co z kolei pozwala na sprawdzenie kodu dotyczącego dokładnie tych elementów. **Druga ikona od lewej pozwala na sprawdzenie responsywności strony** – możemy sprawdzić jak strona będzie prezentować się na ekranach o różnych wymiarach.

Console czyli konsola przeglądarki pozwala m.in. na wykrywanie błędów w kodzie strony, ale również do pisania kodu i testowania, czy kod działa zgodnie z oczekiwaniami.

Pierwszym fragmentem kodu, który początkujący programiści zapisują jest zwykle komunikat „Hello World". Poniższą komendę będziemy zwykle wpisywać w pliku script.js podpiętym do pliku index.html – w ten sposób przeglądarka będzie mogła zinterpretować zapis (console.log() – „wyświetl w konsoli") i wyświetli „Hello World" w konsoli. W tym wypadku jest to zapis poglądowy.

Po wpisaniu komendy console.log('Hello World'); klikamy enter.



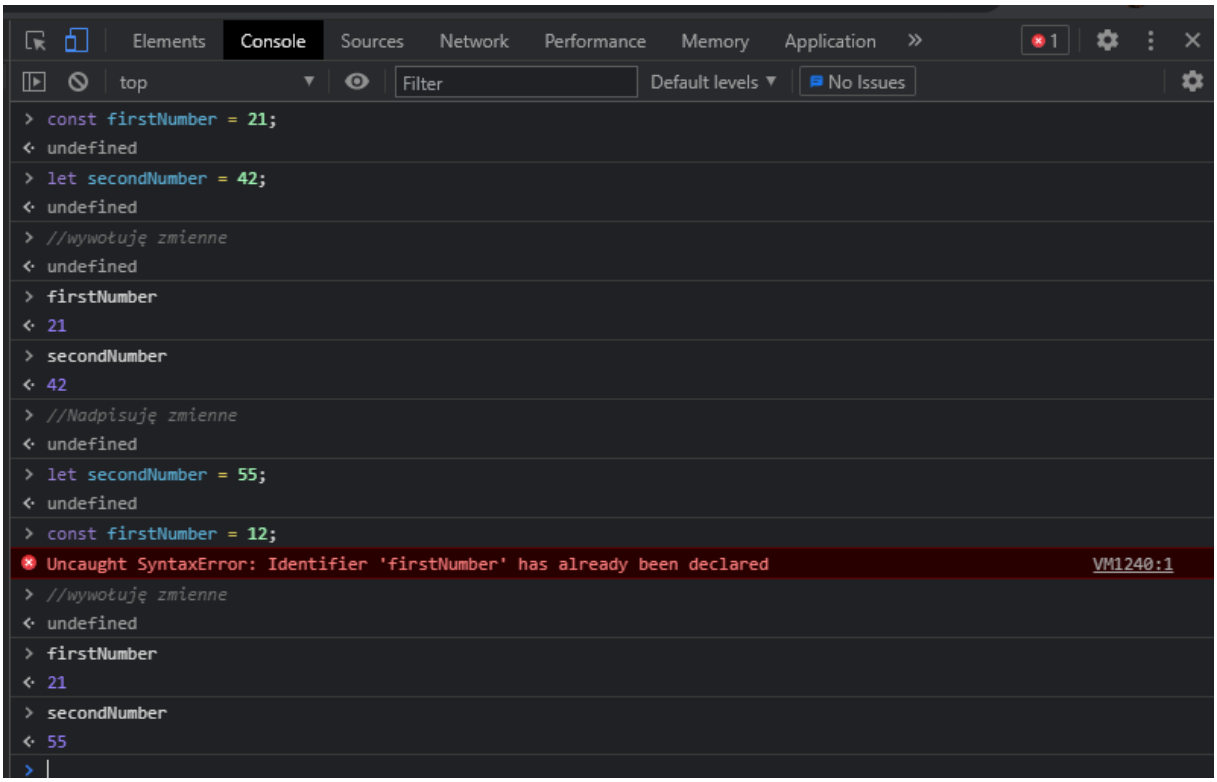
JAVASCRIPT – PODSTAWY

ZMIENNE

Upraszczając, zmienne to „słowa klucze" pod którymi możemy zadeklarować różne wartości, obiekty, teksty itd. Wyróżniamy zmienne **let** oraz **const**. Różnica między nimi polega na tym, że zmienną const deklarujemy tylko jeden raz – nie możemy jej nadpisać. Z kolei zmienna let może zostać nadpisana.

Przed wprowadzeniem tego rozróżnienia można było spotkać zmienną **var**, ale z powodu potencjalnych problemów podczas interpretacji kodu rezygnuje się z jej stosowania.

Zapis poprzedzony dwoma ukośnikami oznacza komentarz – ta część „kodu" nie jest interpretowana przez przeglądarkę.



Jak widać powyżej, zmienna **let** została nadpisana. Pierwotnie przypisano jej wartość 42, następnie 55 – przy powtórnym wywołaniu zmiennej w konsoli pokazała się wartość 55. W przypadku próby nadpisania zmiennej **const** pojawił się komunikat o błędzie i informacja, że zmienna wcześniej została zadeklarowana. Po wywołaniu zmiennej pojawiła się pierwotnie przypisana wartość – 21.

TYPY DANYCH

Number (Liczby)

- let number = 12;
- const numberOne = 15;

String (Ciągi znaków)

- let text = "Jan Kowalski";
- const textName = "Pankracy";

Boolean (Wartości logiczne)

- let trueText = true;
- const falseText = false;

Specjalne

- let noth = null;
- const special = undefined;

Obiekty

- let littleCat = {
  name: "Zeno",
  age: 1.5
  }

Arrow (Tablice)

- const arrow = [1,7,15,"Cat","May"];
- let arr2 = [2,22,222,[2222],2.2];

SPRAWDZANIE TYPU – TYPEOF

OBIEKTY W JAVASCRIPT

Przykładem jest obiekt Math, który pozwala na wykonywanie operacji matematycznych. W przypadku obiektów pełną listę można znaleźć na stronie:

https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Global_Objects

Przykłady:

Math.ceil() – zaokrąglenie w górę;

Math.floor() – zaokrąglenie w dół;

Math.max() – maksymalna wartość ze zbioru;

Math.min() – minimalna wartość ze zbioru;

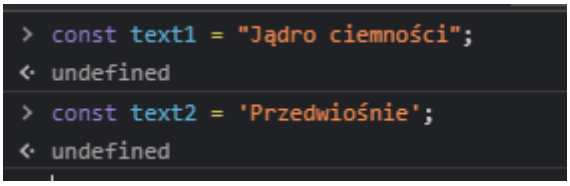
Math.random() – losowa liczba z przedziału 0-1;

Math.round() – zaokrąglanie wartości;

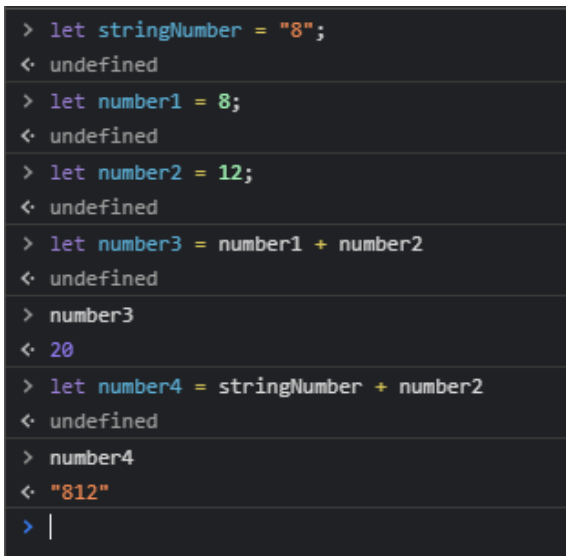
```
> let number1 = 2.22;
undefined
> let number2 = Math.random();
undefined
> number2
0.40706734094459507
> let number3 = Math.round(number2);
undefined
```

STRINGI

Jak wcześniej w obiektach wskazano, w JS można korzystać ze stringów. Stringi zapisujemy w cudzysłowie lub wykorzystując pojedyncze apostrofy:



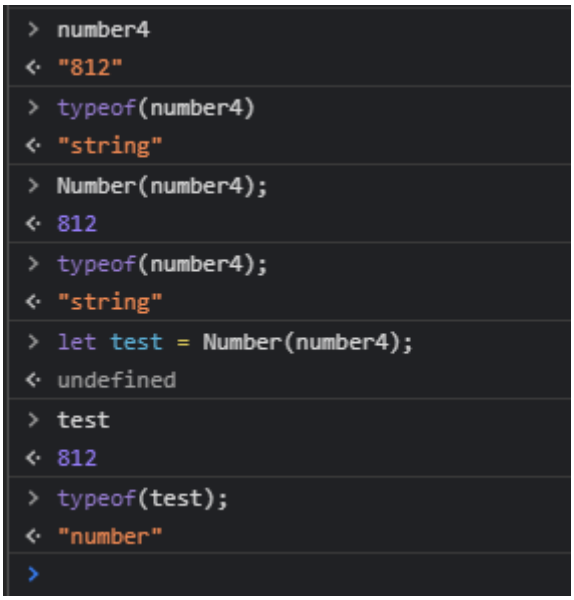
UWAGA! String ≠ Liczba



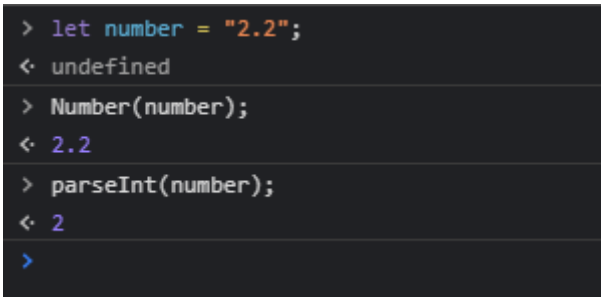
Jak widać powyżej, nie możemy wykonać działań matematycznych pomiędzy stringiem a liczbą. W takim wypadku liczba zostaje przyłączona do stringa, a uzyskana całość stanowi string.

Funkcje i stringi

String może zostać przekształcony do typu number za pomocą funkcji **Number();.** Funkcja może konwertować również wartości logiczne i obiekty. Zmienne zostały zadeklarowane na screenie wcześniej.



Podobnie działa funkcja parseInt();, która zamienia string na liczbę. Różnica polega na tym, że parseInt(); umożliwia zamianę stringa na liczbę całkowitą, z kolei Number() – na dowolną liczbę.



STRINGI I METODY

Przykładowe metody:

string.toUpperCase(); - konwertuje string na duże litery;

string.toLowerCase(); - konwertuje string na małe litery;

string.charAt(index); - zwraca określony znak z łańcucha znaków; (index: liczba całkowita od 0 do liczby o 1 mniejszej od długości łańcucha);

str.concat(string2, string3, ..., stringN); - łączy tekst dówch lub więcej łańcuchów znaków (np. stringów) i zwraca nowy łańcuch;

string,trim(); - usuwa białe znaki z poczatku i końca stringa;

UWAGA! Długość łańcucha znaków można określić za pomocą metody **string.length;**

```
> let text1 = "Przypadki Robinsona Kruzoe";
< undefined
> let text2 = "    Makbet     ";
< undefined
> text1
< "Przypadki Robinsona Kruzoe"
> text2
< "    Makbet     "
> text1.length
< 26
> text1.toUpperCase();
< "PRZYPADKI ROBINSONA KRUZOE"
> text2.toLowerCase();
< "    makbet     "
> text1.charAt(2);
< "z"
> text1.charAt(text1.length - 24);
< "z"
> text1.concat(text2,"KOT","_","pies");
< "Przypadki Robinsona Kruzoe    Makbet     KOT_pies"
> text2.trim();
< "Makbet"
>
```

OPERATORY

OPERATORY PRZYPISANIA (ARYTMETYCZNE)

Operatory, które służą do przypisania do zmiennej jakiejś wartości, obiektu, itd.

```
> let x = 4;
< undefined
> let y = 8;
< undefined
> //dodawanie
< undefined
> x + y;
< 12
> //odejmowanie
< undefined
> x - y;
< -4
> //dzielenie
< undefined
> x / y;
< 0.5
> //mnożenie
< undefined
> x * y;
< 32
> //reszta z dzielenia - modulo
< undefined
> x % y;
< 4
> //Inkrementacja czyli x = x + 1;
< undefined
> x++;
< 4
> x
< 5
> //Dekrementacja czyli y = y - 1;
< undefined
> y--;
< 8
> y
< 7
```

W przypadku dekrementacji czy inkrementacji możemy również wykorzystać zapis:

```
> let x = 4;
< undefined
> let y = 8;
< undefined
> x++;
< 4
> x
< 5
> x += 1;
< 6
> //Jeśli chcemy zwiększać każdorazowo wartość np. o 5:
< undefined
> x += 5;
< 11
> y--;
< 8
> y;
< 7
> y -= 1;
< 6
> y -= 12;
< -6
>
```

```
> //kiedy zamiast dwóch liczb mamy string
< undefined
> let x = "2";
< undefined
> let y = 4;
< undefined
> x + y;
< "24"
> x - y;
< -2
> x / y;
< 0.5
> x * y;
< 8
> x % y;
< 2
> //nie licząc dodawania - JS podczas wykonywania działań zamienia
  string na liczbę (oczywiście, jeśli jest taka możliwość. W przypadku
  stringa "2koty" nie będzie to możliwe).
< undefined
>
```

OPERATORY PORÓWNANIA

Stosuje się je zwykle w instrukcjach warunkowych. Jak nazwa wskazuje – służą do porównywania obu stron rónwnia, a zwracana wartość to true lub false.

UWAGA!

- **==** ⟶ porównywana jest wartość zmiennych;
- **!=** ⟶ wykrzyknik wskazuje na zaprzeczenie, tutaj: „nie są równe”;
- **===** ⟶ porównywana jest wartość i typ (aby konsola zwróciła „true” oba warunki muszą być spełnione);
- **!==** ⟶ „typ lub wartość nie są równe” (aby konsola zwróciła „true” przynajmniej jeden warunek musi być spełniony).

```
> let x = 22;
< undefined
> let y = 48;
< undefined
> x == y;
< false
> x != y;
< true
> x === y;
< false
> x !== y;
< true
> x > y;
< false
> x < y;
< true
> x >= y;
< false
> x <= y;
< true
> //jeśli pojawi się np. string
< undefined
> let text = "22";
< undefined
> text == x;
< true
> text === x;
< false
> text !== x;
< true
>
```

OPERATORY LOGICZNE

Operatory logiczne stosowane są w przypadku instrukcji warunkowych.

- **&& czyli AND (i)** ⟶ jeśli pierwszy warunek nie jest spełniony, to reszta nie jest sprawdzana, a w konsoli zwracana jest wartość false;
- **|| czyli OR (lub)** ⟶ jeśli chociaż jeden warunek jest spełniony, to w konsoli zwracana jest wartość true;
- **! czyli NOT (nie)** ⟶ jeśli warunek jest prawdą, zwrócone zostanie false, jeśli nie jest prawdą – zwrócone zostanie true;
- **^ czyli OR (albo)** ⟶ tylko jeden z warunków powinien być prawdziwy, wtedy zwracane jest true, jeśli oba są prawdziwe lub nieprawdziwe – wtedy zwracane jest false.

```
> let x = 22;
< undefined
> let y = 48;
< undefined
> (x != 22) && (x > 15);
< false
> (x != 22) || (x > 15);
< true
> !(y > 20);
< false
> (y >30) ^ (y != 48);
< 1
> //w powyższym przypadku wartość 1 = true, z kolei 0 oznaczałoby false
< undefined
> (y >30) ^ (y == 48);
< 0
>
```

OPERATORY WARUNKOWE

Operatory warunkowe stosowane są zwykle w instrukcjach warunkowych.

Zapisujemy zgodnie ze wzorem:

WARUNEK ? "KOMUNIKAT JEŚLI PRAWDA" : "KOMUNIKAT JEŚLI FAŁSZ";

```
> let number = 25;
< undefined
> let result = (number > 2) ? "prawda" : "fałsz";
< undefined
> result
< "prawda"
> let name = "Hellen";
< undefined
> let resultName = (name.length == 6) ? "Tak, to prawda" : "Nie, to nieprawda";
< undefined
> resultName
< "Tak, to prawda"
> //czyli: czy długość stringa/ilość znaków w łańcuchu zmiennej name ("Hellen") to 6
  znaków? Jeśli tak, to konsola zwróci "Tak, to prawda", zaś jeżeli nie jest to
  prawdą to zwróci komunikat "Nie, to nieprawda"
< undefined
>
```

INSTRUKCJE WARUNKOWE – IF

```
> let catFood = "pusta miska";
< undefined
> if (catFood === "pusta miska") {
      console.log("Nakarm kota!");
  } else if (catFood === "pełna miska") {
      console.log("Kot nie będzie głodny");
  } else {
      console.log("Kot wyruszył na polowanie");
  }
  Nakarm kota!
< undefined
> let catFood = "pełna miska";
< undefined
> if (catFood === "pusta miska") {
      console.log("Nakarm kota!");
  } else if (catFood === "pełna miska") {
      console.log("Kot nie będzie głodny");
  } else {
      console.log("Kot wyruszył na polowanie");
  }
  Kot nie będzie głodny
< undefined
```

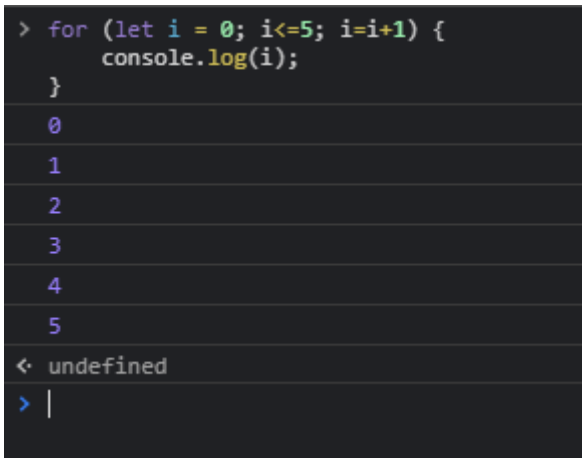
```
> let catFood = "dzisiaj sardynki";
< undefined
> if (catFood === "pusta miska") {
      console.log("Nakarm kota!");
  } else if (catFood === "pełna miska") {
      console.log("Kot nie będzie głodny");
  } else {
      console.log("Kot wyruszył na polowanie");
  }
  Kot wyruszył na polowanie
```

INSTRUKCJE WARUNKOWE – SWITCH

```
> let catFood = "pusta miska";
< undefined
> switch(catFood) {
      case "pusta miska": {
      console.log("Nakarm kota!");
      break;
      }
      case "pełna miska": {
      console.log("Kot nie będzie głodny");
      break;
      }
      default: {
      console.log("Kot jest najedzony, więc jest niewzruszony");
      }
  }
  Nakarm kota!
< undefined
> let catFood = "dzisiaj sardynki";
< undefined
> switch(catFood) {
      case "pusta miska": {
      console.log("Nakarm kota!");
      break;
      }
      case "pełna miska": {
      console.log("Kot nie będzie głodny");
      break;
      }
      default: {
      console.log("Kot jest najedzony, więc jest niewzruszony");
      }
  }
  Kot jest najedzony, więc jest niewzruszony
< undefined
>
```

PĘTLA FOR

Chcemy, aby w konsoli pojawiły się cyfry od 0 do 5. (Wiemy, w jakim zbiorze będziemy działać).



UWAGA! Zamiast zapisu i=i+1 można tutaj zastosować i++;

PĘTLA WHILE

Stosujemy zwykle wtedy, kiedy nie wiemy w jakim zakresie/zbiorze będziemy działać.

W tym wypadku, jeśli losowa wartość „i" będzie różna od 274 i mniejsza od 7 to pętla wyświetli napis „Sukces!".