

# Функции и асинхронность в JavaScript



[ORTDNIPRO.ORG/JS](https://ORTDNIPRO.ORG/JS)

# 1. Функции и функциональные выражения

# Функции в JavaScript

```
2
3  function action(a, b, c){
4      |   let sum = a + b + c;
5      |   return sum;
6      | }
7
8  let process = function(a, b, c){
9      |   let sum = a + b + c;
10     |   return sum;
11     | }
12
13  let calculate = (a, b, c) => a + b + c;
14
15  typeof action; //function
16  typeof process; //function
17  typeof calculate; //function
18
```

Функции в JavaScript – блоки кода которые возможно вызывать (выполнять) многократно. Синтаксисом JS предусмотрено несколько способов определения функций: Объявление функции (***Function Declaration***) (3), Функциональное выражение (***Function Expression***, она же «анонимная» функция) (8), и стрелочные-функции (***arrow-function***, они же лямбда-функции) (13). Функции в JavaScript – тип данных, функцию мы можем размещать в переменных, как и другие типы данных. Отличие в том, что функции мы можем вызывать.

Подробнее: <https://learn.javascript.ru/function-basics>

Подробнее: <https://learn.javascript.ru/arrow-functions-basics>

# Оператор ... и функции

```
2
3   let process = function(a, b, c, ...others){
4       console.log(others);
5       let sum = a + b + c;
6       return sum;
7   }
8
9   process(1,2,3,4,5,6,7); // return 6;
10  // in console: [4,5,6,7];
11
```

Функция может принимать параметры и возвращать результат своей работы для дальнейшего использования (оператор *return*).

Но при помощи оператора ... (в данном случае его называют *rest-оператором*) мы можем принять любое количество параметров и работать с ними как с массивом (**ES2015**).

Подробнее: <https://learn.javascript.ru/rest-parameters-spread-operator>

## 2. Таймеры в JavaScript

# Таймеры в JavaScript

```
2
3   let f1 = function(){
4       |   console.log("Function for Timeout called");
5       |   }
6
7   let f2 = function(){
8       |   console.log("Function for Interval called");
9       |   }
10
11   let timeout_id = setTimeout(f1, 1000);
12
13   let interval_id = setInterval(f2, 3000);
14
```

**setTimeout**(*some\_function*, *delay*) – вызовет функцию *some\_function* через *delay* миллисекунд. Сделает это один раз.

**setInterval**(*some\_function*, *delay*) – вызовет функцию *some\_function* через *delay* миллисекунд. И будет повторять вызов каждые *delay* миллисекунд.

Обе функции возвращают **id** таймера, с помощью которого и функций **clearTimeout(id)** и **clearInterval(id)** уничтожить таймер еще до его вызова. Обе функции можно отнести к инструментам **асинхронности**.

Подробнее: <https://learn.javascript.ru/settimeout-setinterval>

### 3. Геолокация и **callback**'и

# Геолокация в теории



Широта == Latitude

Долгота == Longitude

```
{ ..., latitude: 48.4767, longitude: 35.0543, ... };
```



# Геолокация на практике

```
2
3 // 'Classic' version
4 navigator.geolocation.getCurrentPosition( position => {
5     console.log('Your position: ', position.coords);
6 }, error => {
7     console.log('Geolocation error:', error);
8 })
9
```

У браузера есть возможность узнать координаты пользователя на местности. Для этого мы можем воспользоваться методом **navigator.geolocation.getCurrentPosition()** который принимает **callback** функции для получения координат и информации об ошибке. Но важно **проверять поддерживает ли браузер геолокацию** проверяя наличие свойства **geolocation** объекта **navigator**.

Подробнее: <https://developer.mozilla.org/ru/docs/Web/API/Geolocation/getCurrentPosition>

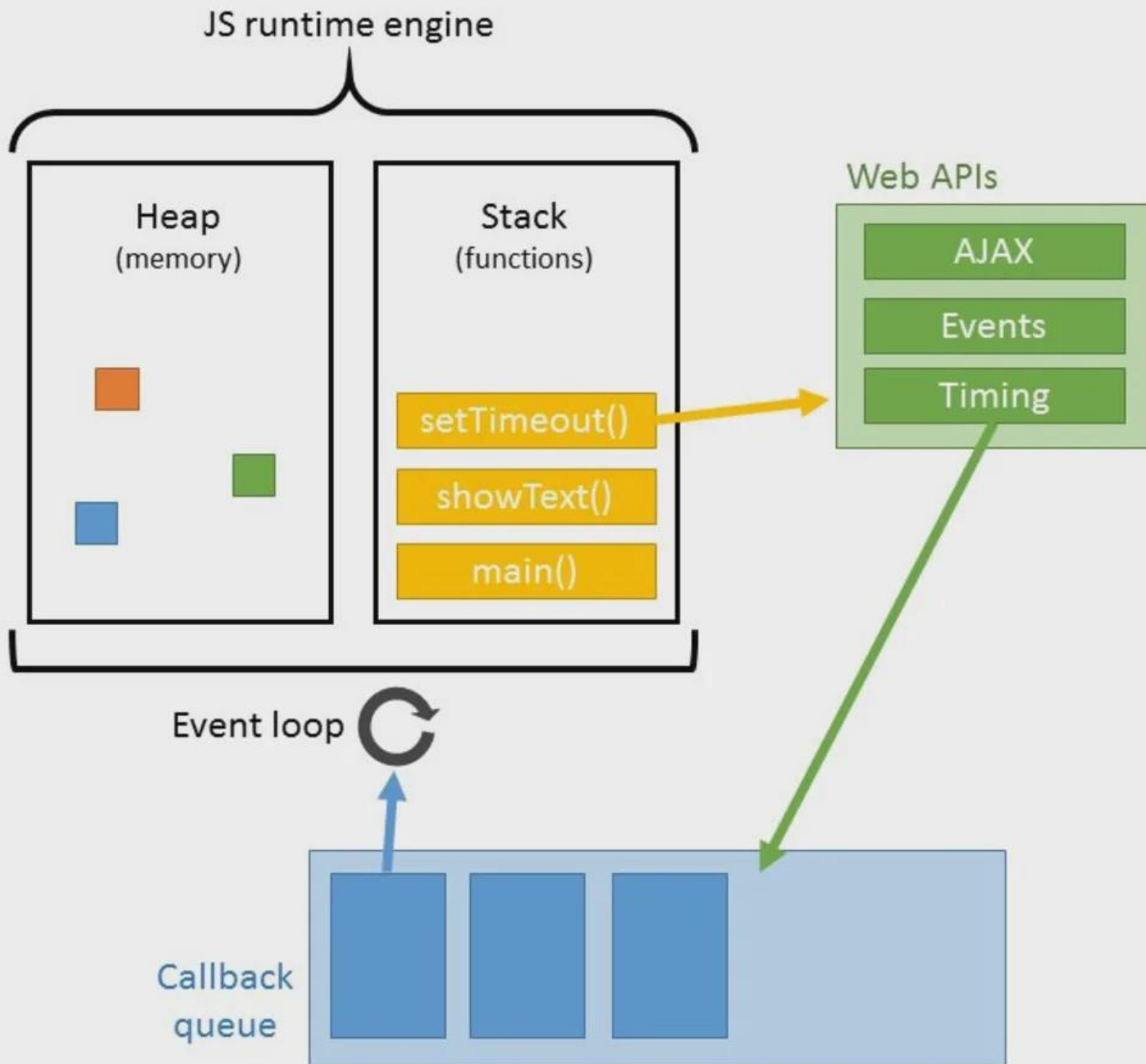
# Немного о статических карта на примере Here Map

```
https://image.maps.api.here.com/mia/1.6/mapview?app_id=oZmMWRV4tAjQmgkxBvF0&app_code=x5pKHqifhw1mnS_zBTIFsA&z=11&w=600&h=600&c=48.4608,35.0501
```

Сервис **Here Map** предоставляет возможность размещать на наших страницах картографические материалы, управляя позицией и масштабом отображения.

Вы можете воспользоваться шаблоном в репозитории [./src/template-geolocation/](#)

## 4. Цикл событий / Event Loop

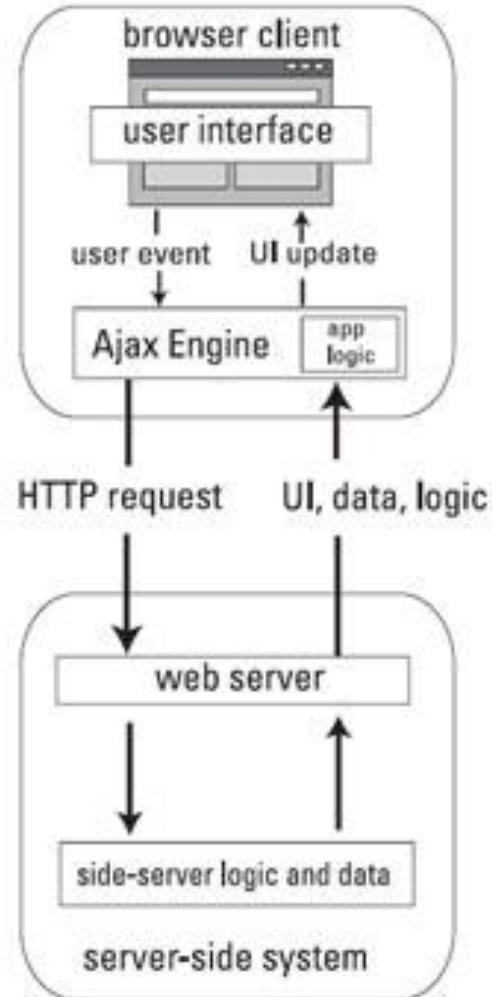
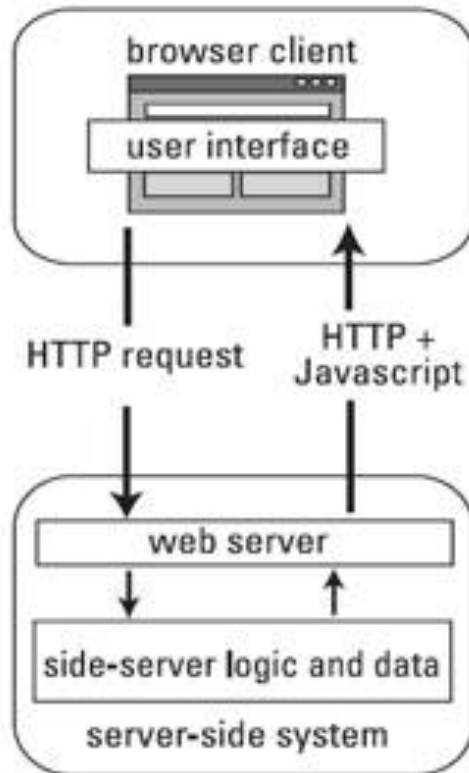


# Event Loop

**JavaScript** однопоточный язык программирования, но тем не менее нам доступны асинхронные инструменты. Доступны они за счёт функционирования механизма **Event Loop** (или **цикла событий**, но **не стоит путать с событиями DOM**).

## 5. **AJAX** и функция **fetch()**

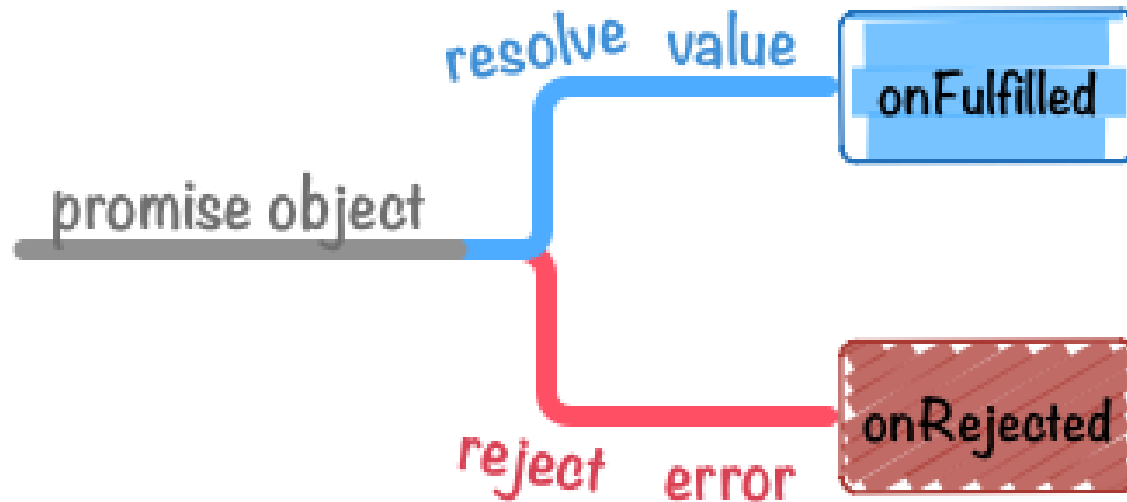
# Asynchronous JavaScript And XML



Идея заложенная в **AJAX** – не перезагружая страницу, запросить (или передать) у сервера новые данные и использовать их в документе. Для выполнения запросов нам доступна функция **fetch()**.

## 6. Операторы `async/await`

# Объект Promise



**Promise** – механизм позволяющий писать асинхронный код последовательно (насколько это возможно), избегая вложенности **callback**’ов. **Promise** – объект который принимает функцию, в которой запускается асинхронная операция, при помощи параметров функции есть возможность из асинхронного кода сообщить об успешном или не успешном завершении операции.

Подробнее: <https://learn.javascript.ru/promise/>



# async/await – упрощение кода Promise'ов

```
2  
3 let url = 'https://bank.gov.ua/NBUStatService/  
4           v1/statdirectory/exchange?json';  
5  
6  
7  
8  
9  
10  
11  
12  
13
```

```
let result = await fetch(url);  
result     = await result.json();  
  
console.log(result);
```

**async/await** – надстройка над **Promise** позволяющая писать код в полностью привычном синхронном стиле, при этом откладывая ожидания завершения операций до тех пор пока её результат действительно понадобится;

**async** – отмечает функцию как асинхронную (результат такой функции оборачивается в **Promise**);

**await** – при вызове асинхронных функций указывает, что не нужно ждать результата сейчас

Подробнее: <https://learn.javascript.ru/async-await>

# 7. WebAPI

# API Национального Банка Украины



**НАЦІОНАЛЬНИЙ  
БАНК УКРАЇНИ**

Валютные API, информация о финансовом рынке и банковском секторе

<https://bank.gov.ua/ua/open-data/api-dev>

## WebAPI построенные на обмене данными в формате JSON

Разработчикам доступно огромное количество сервисов которые предоставляющие доступ к данным в формате **JSON**. Такого рода сервисы носят название **WebAPI**.



```
[{"ccy": "USD", "base_ccy": "UAH", "buy": "28.05000", "sale": "28.25000"},  
{"ccy": "EUR", "base_ccy": "UAH", "buy": "31.95000", "sale": "32.45000"},  
{"ccy": "RUR", "base_ccy": "UAH", "buy": "0.41500", "sale": "0.43500"},  
{"ccy": "BTC", "base_ccy": "USD", "buy": "6143.7724", "sale": "6790.4852"}]
```

<https://api.privatbank.ua/>

# JSON (JavaScript Object Notation)

**JSON** - текстовый формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Основан на синтаксисе (правилах записи) массивов в **JavaScript**. Формат поддерживается практически во всех современных языках программирования.

```
[ { "name": "Jane",  "age": 23 },  
  { "name": "Max",   "age": 16 },  
  { "name": "Maria", "age": 34 },  
  { "name": "Alex",  "age": 20 },  
  { "name": "Cate",  "age": 45 } ]
```

<http://www.json.org/json-ru.html>

Для работы с форматом **JSON** у нас есть два метода: **JSON.stringify(data)** – который преобразует структуру данных в строковое представление, и метод **JSON.parse(str)** который делает обратное действие.

Подробнее: <https://learn.javascript.ru/json>

# 8. Перебирающие методы массивов

# Метод .sort() и функция-компаратор

```
2
3   let arr = [23, 4, 67, 117, 34, 0, 55, 78, 5, 9];
4
5   arr.sort(function(a, b){
6       if(a > b){
7           return 1;
8       }else if(a < b){
9           return -1;
10      }else{
11          return 0;
12      }
13  });
14  //arr.sort((a,b) => a - b);
15
16  console.log(arr);
17  //[0, 4, 5, 9, 23, 34, 55, 67, 78, 117]
18
```

Методу **.sort()** массивов можно передать функцию (т.н. функцию-компаратор) которая «подскажет» браузеру как сравнивать два элемента между собой. Функция принимает 2 элемента и должна вернуть 0 если они равны, отрицательное число если второй элемент больше или положительное если первый элемент больше.

Подробнее: <https://learn.javascript.ru/array-methods>

# Полезные методы преобразования массивов

**.filter();**

Метод **.filter()** формирует новый массив занося в него элементы из старого, но только те которые «одобрит» функция переданная методу в качестве параметра.

**.map();**

Метод **.map()** формирует новый массив занося в него элементы из старого, но предварительно пропуская каждый элемент через функцию переданную методу в качестве параметра. Эта функция может любым образом преобразовать элемент.

**.reduce();**

Метод **.reduce()** позволяет хранить при переборе элементов какое-либо промежуточное значение, оно передаётся в первом параметре функции (передаваемой методу). При каждом вызове то что возвращает функция становится этим самым «промежуточным» значением для следующего вызова функции. В результате **.reduce()** возвращает самое последнее «промежуточное значение»

Подробнее: <https://learn.javascript.ru/array-methods#preobrazovanie-massiva>



**Будет полезным**

# Перебирающие методы

В **JavaScript** есть еще ряд методов массивов, а именно: **.every()**, **.some()**, **.find()**, **.findIndex()**, **.forEach()** узнайте чем они могут быть полезны.

**На следующем занятии**

**На следующем занятии**

**Принципы и подходы ООП в JavaScript и  
всё, что с этим связано...**

**Домашнее задание**  
**/сделать**

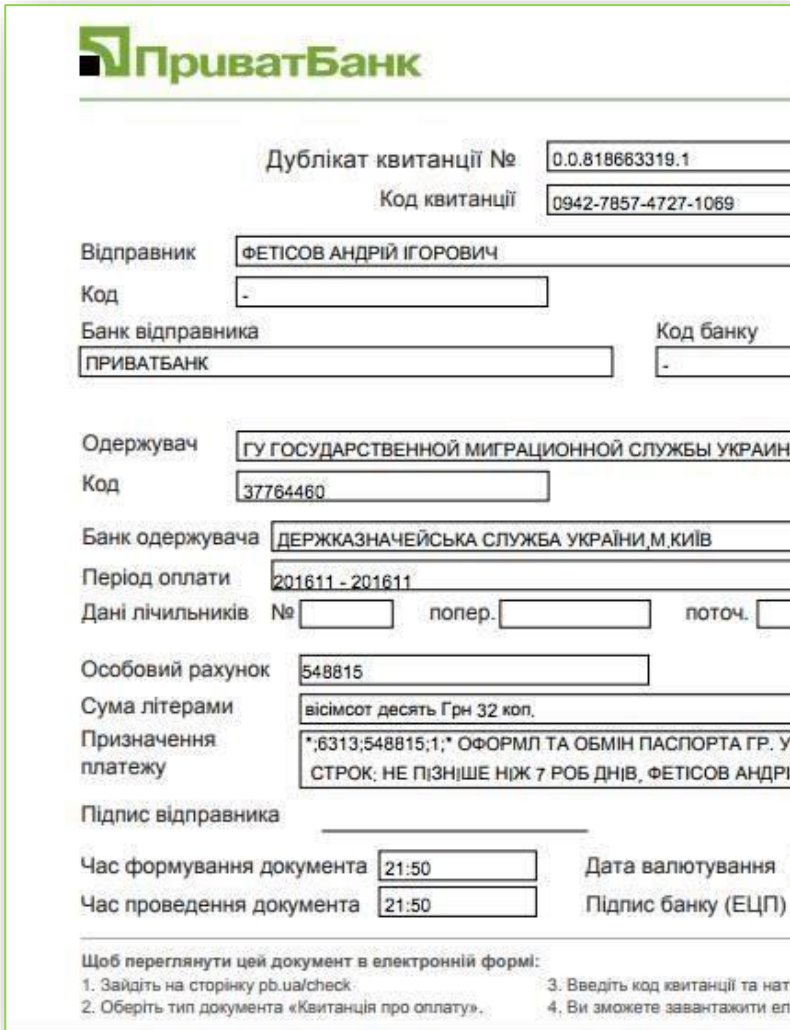
# Домашнее задание #B.1

«Азбука пилотов» (или официально **фонетический алфавит ICAO**) - стандартизированный способ прочтения букв алфавита английского языка в авиации. Каждая буква кодируется словом, которое при плохой связи позволяет с высокой вероятностью распознать букву которая передаётся. Ваша задача, написать скрипт, который будет переводить буквенно-цифровую комбинацию в набор слов из «азбуки пилотов».

**Например:** пользователь вводит комбинацию буквенно-цифровую, (буквы **только латинские**) (например: **KL1386**), а скрипт выдает «расшифровку» в **соответствии с алфавитом ICAO** (например: **Kilo Lima One Three Eight Six**). Регистр вводимой комбинации не должен влиять на результат (т.е. большие и маленькие буквы дают один и тот же результат).



## Домашнее задание #B.2



**ПриватБанк**

Дублікат квитанції № 0.0.818663319.1  
Код квитанції 0942-7857-4727-1069

Відправник ФЕТИСОВ АНДРІЙ ІГОРОВИЧ  
Код -  
Банк відправника ПРИВАТБАНК Код банку -

Одержувач ГУ ГОСУДАРСТВЕННОЙ МИГРАЦИОННОЙ СЛУЖБЫ УКРАИНЫ  
Код 37764460  
Банк одержувача ДЕРЖКАЗНАЧЕЙСЬКА СЛУЖБА УКРАЇНИ, М. КИЇВ

Період оплати 201611 - 201611  
Дані лічильників № попер. поточ.

Особовий рахунок 548815  
Сума літерами вісімсот десять грн 32 коп.  
Призначення платежу \*6313;548815;1;\* ОФОРМЛ ТА ОБМІН ПАСПОРТА ГР. У  
СТРОК: НЕ ПІЗНІШЕ НІЖ 7 РОБ ДНІВ, ФЕТИСОВ АНДРІ

Підпис відправника  
Час формування документа 21:50 Дата валютування  
Час проведення документа 21:50 Підпис банку (ЕЦП)

Щоб переглянути цей документ в електронній формі:  
1. Зайдіть на сторінку rb.ua/check. 3. Введіть код квитанції та нат  
2. Оберіть тип документа «Квитанція про оплату». 4. Ви зможете завантажити ел

Написать скрипт которые будет словами записывать сумму заданную числом которое ввёл пользователь в пределах от 1 до 999 (включительно). Например **643** => «**шістьсот сорок три гривны**» (не забывая добавлять слово *гривен, гривна* и т.д. в зависимости от необходимого склонения).