1. Webシステム基礎実験（JavaScritpクラス）

1.1. 環境設定編

1.1.1. 利用するクラウドサービス

本演習では，以下のクラウドサービスを使用する．そこで，まずはアカウントを作成する．なお，Paiza Cloudは，Githubアカウントで利用できるので，そちらを利用しても構わない．

1. Github https://github.io/
2. Paiza Cloud https://paiza.cloud/

1.1.2. Paiza Cloudの使い方

上記に記したPaiza CloudのURLにWebブラウザからアクセスすると，画面上部に「ログイン」というボタンが有るのでクリック（タップ）する．すると，作成したアカウントとパスワードを入力する画面が現れる．下の方に，別のサービスのアカウントを利用してログインするボタンが有るので，「Githubログイン」をクリック（タップ）する．

すると，Githubのユーザ名（またはメールアドレス）とパスワード入力画面になるので，入力して「Sign in」する．

1.1.3. Paiza Cloudでのサーバの作成

Sign inすると，画面中央に「新規サーバ作成」ボタンが現れるので，作成する．その際，サーバ名はそのまま利用して良い． その下の初期インストールするものの中で，Web開発の中の「node.js」を選択し，「新規サーバ作成」を選ぶ 後々はデータベースの中の「MySQL」も利用するが，最初は「Node.js」のみで良い．

1.1.4. Gitのユーザ名とメールアドレスを設定する

gitサーバにアップロードする際に必要なので，最初にやっておくと良い． この作業は，サーバを作り直すたびに毎回実行するものと思って欲しい．

```
~$ git config --global user.name "Hiroshi Suda"
~$ git config --global user.email "suda@net.it-chiba.ac.jp"
```

以下のように確認してみよう

```
~$ git config --list
user.name=Hiroshi Suda
user.email=suda@net.it-chiba.ac.jp
```

1.1.5. Githubからプログラムをダウンロードする

Paiza Cloudの画面内の「ターミナル」を開く．すると，Linuxのコマンド入力画面が現れる．ここで，gitコマンドを利用してファイルをダウンロードする．以下の最初の行を入力する．

```
~$ git clone https://github.com/sudahiroshi/websystem.git
Cloning into 'websystem'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
Unpacking objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 5 (delta 1), pack-reused 0
~$
```

続いて，カレントディレクトリを変更し，プログラムを起動する．

```
~$ cd websystem
~/websystem$ node server1.js
```

ここでエラーが表示されてプログラムが停止する場合は，以下のように「sudo」を付けて実行すると良い．

```
~/websystem$ sudo node server1.js
```

すると左端にブラウザアイコンが追加されるので， クリックするとブラウザを経由してプログラムの出力を確認できる．

1.1.6. リポジトリのFork

各自のリポジトリを作成して，gitコマンドに習熟する． いきなり新規リポジトリを作成する前に，Github上でFork機能を利用する． Webブラウザで，websystemを開いて，右上の方に有るForkをクリックすると，そのリポジトリが自分の管理下にコピーされる． Forkすると，画面左上にある「ユーザ名/リポジトリ名」のユーザ名が自分のものになっているはずである． なお，「Clone or download」で出てくるURLも変更されている． このURLは後で使用する．

ForkされたリポジトリをPaiza Cloud上で使用するのだが，同じ名前のリポジトリが有ると不具合が起こる可能性があるので，一旦動作しているサーバ破棄する． Paiza Cloudの画面左上にある青いバー（「新規ファイル」の上）をクリックするとメニューが出てくるので，サーバの削除を選択する． 本当に削除して良いか確認画面が出るので，削除する．

続いて，再度「新規サーバ作成」をクリックして起動する． とりあえずNode.jsを選択して，新規サーバ作成をクリックする． 起動したらターミナルを開いて，gitの設定を行う． なお，内容があっていることを確認すること．

それでは，gitコマンドを使って手元にcloneし，サーバを起動しよう．

```
~$ git clone <ここに各自のリポジトリをcloneするためのURLを入れる>
~$ cd websystem
```

```
~/websystem$ sudo node server1.js
```

Paiza内のWebブラウザで動作を確認できる.

1.1.7. ファイルを変更し，Githubにアップロードする

確認用に動かしたサーバは「ctrl+c」で停止しておく． ここでは，server2.jsを変更しよう． server2.jsの6～11行目に，「res.write」が並んでいるが，ここではHTMLが直接書かれている． この内容を好きなように変更して，保存する.

続いて，以下のように順次入力し，アップロードする． **gitの設定が済んでいないと「git config」しなさいとのエラーが表示される.**

なお，「git commit」をする際のコメントは，あとで分かるような変更箇所や変更理由などを書く項目なので，そのまま入力しないこと.

```
~/websystem$ git commit -am 'ここには変更箇所などを書く'
[master b9b0945] ここには変更箇所などを書く
 1 file changed, 3 insertions(+)
~/websystem$ git push
Username for 'https://github.com': sudahiroshi   ←ユーザ名は各自のものを入力する
Password for 'https://sudahiroshi@github.com':    ←表示されていないが，パスワードを入力する
Counting objects: 3, done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/sudahiroshi/webssystem.git
   8e04f1b..b9b0945  master -> master
```

「git push」がうまく動作していることを，githubで確認しよう． ファイル名の横に，入力したコメント（変更箇所など）が表示され，ファイルの内容が新しくなっていれば良い.

1.1.8. 動的なWebページを作る

編集方法とアップロードまでできたので，次に動的なWebページを作成してみよう． ここで言う「動的」とは，サーバのプログラム内で何か処理を行ってクライアントに返すことである． （参考までに通常のWebページは「静的」である）

動的な動作をするWebページのプログラムを以下に示す． ここで，5行目（server.onの次の行）では，現在時刻を取得し，変数nowに代入している． その後，12行目でnowを表示している.

```
var http = require('http');
var server =http.createServer();

server.on( 'request', function(req,res) {
    var now = new Date().getTime();
    res.writeHead( 200, {'Content-Type' : 'text/html' });
    res.write('<!DOCTYPE html>');
```

```
    res.write('<html lang=ja>');
    res.write('<head><meta charset="UTF-8"></head>');
    res.write('<body>');
    res.write('<h1>Hello world</h1>');
    res.write('現在の時刻は'+now);
    res.write('</body>');
    res.write('</html>');
    res.end();
});

server.listen(80);
```

それでは実際に動かして確認してみよう． もう動かし方は覚えたと思うので，ファイル名が`server3.js`であることだけを記しておく． なお，現在時刻はきちんと整形しなければ，時刻として読めない数字が表示されるのみである． 今回は，数字が表示されれば良いものとする．

1.1.9. ファイル名によって異なるページを返す

通常，Webページは多くのHTMLファイルから構成される． しかし，上で紹介したWebサーバは，決まった内容しか返すことができない． そこで，複数のWebページに対応したサーバに拡張していく．

まずは，プログラム中でURL（ファイル名などの表示したい情報）の取得方法を確認しよう． 以下が，URLをターミナルに表示するプログラムである． （なお，これまでvarと書いていた箇所がconstやletに変更されている．これについては追い追い説明する）

2行目が，URLを取得するためのライブラリを使用するための宣言である． そして，6行目でURLを取得し，13行目でターミナルに表示している． （console.logはターミナルに表示するためのメソッド）

```
const http = require('http');
const url = require('url');
const server =http.createServer();

server.on( 'request', function(req,res) {
    let url_parse = url.parse(req.url,true);
    res.writeHead( 200, {'Content-Type' : 'text/html' });
    res.write('<!DOCTYPE html>');
    res.write('<html lang=ja>');
    res.write('<head><meta charset="UTF-8"></head>');
    res.write('<body>');
    res.write('<h1>Hello world</h1>');
    console.log(url_parse);
    res.write('</body>');
    res.write('</html>');
    res.end();
});

server.listen(80);
```

プログラムの流れを理解したら，動作確認をしてみよう． ファイル名は`server4.js`である． Webブラウザからアクセスすると，ターミナル上に以下の内容が表示される． この中の，pathnameやpathがファイル名に相当する．

```
Url {
  protocol: null,
  slashes: null,
  auth: null,
  host: null,
  port: null,
  hostname: null,
  hash: null,
  search: null,
  query: {},
  pathname: '/',
  path: '/',
  href: '/' }
```

さて，続いてWebブラウザのURL欄を見てみよう． 渡しの場合は`https://suda-hiroshi-1.paiza-user.cloud/`というURLが入力されている． ここで，最後の`/`の後ろに`test.html`などと付けて，server4.jsの出力を見てみよう． すると，pathnameなどに変化が現れているはずである．

よって，if文やswitch文を使って，pathnameに一致する内容を返すプログラムに変更すれば良いことになる．