Jupyter Notebook 上の教材作成について

山口大学教育学部　北本卓也

1　はじめに

近年では、ブラウザ上で動作するアプリケーションの人気が高まっており、多くアプリケーションが作成されつつある。教材作成においても、ブラウザ上で動作するものが増えており、筆者もこれまで、参考文献 [1],[2],[3] でブラウザ上で学習教材を JavaScript で作成する試みについて研究・開発を行ってきた。

本稿では、最近人気を高めつつある「Jupyter Notebook」に着目する。Jupyter Notebook は Python などを Web ブラウザ上で記述・実行できる統合開発環境だが、オープンソースのフリーソフトウェアであり，近年では様々なプログラミング言語をサポートし、広く使われている。この Jupyter Notebook はブラウザ上で動作するため、JavaScript のプログラムを組み込むことにより、HTML ファイルを操作し、見た目や動作を変えることができる。本稿では JavaScript を用いた Jupyter Notebook 上の教材作成について述べる。

2　Jupyter Notebook について

2.1　Jupyter Notebook とは

Jupyter Notebook は様々なプログラミング言語 (ただし Python が使われる場合が多い) に対応した統合開発環境の１つである。ブラウザ上で動作する Web アプリケーションであるため、ブラウザが動作する環境であればどこでも使うことができる。解説の文章、図とプログラムを１つのファイルに混在させることが可能であり、Jupyter Notebook の画面は従来のプログラムのソースコードとは少し見た目が異なる (図 1 を参照)。図 1 で「プログラムの部分」と書かれている枠で囲まれた灰色の部分 (この部分を「セル」と呼ぶ) に Python の命令 (プログラム) を入力して実行させる。

Jupyter Notebook は広く使われているため公開されているライブラリが多くあり、科学技術計算やデータ解析、機械学習を行うライブラリも豊富に用意されている。特に統計解析やディープラーニングの分野で人気が高い。

2.2　magic コマンド

Jupyter Notebook は統合開発環境なので、ベースとなっているプログラミング言語 (多くの場合は Python) の文法に沿った命令を入力する必要があるが、「magic コマンド」という特別な命令を使うとシステムを直接操作をしたり、他のプログラミング言語を使ったりすることが可能になる。

`%lsmagic` で利用可能な magic コマンドの一覧を得ることが可能であり、図 2 にそれを示すが、本稿では特にその中の１つ `%%html` を用いていく。

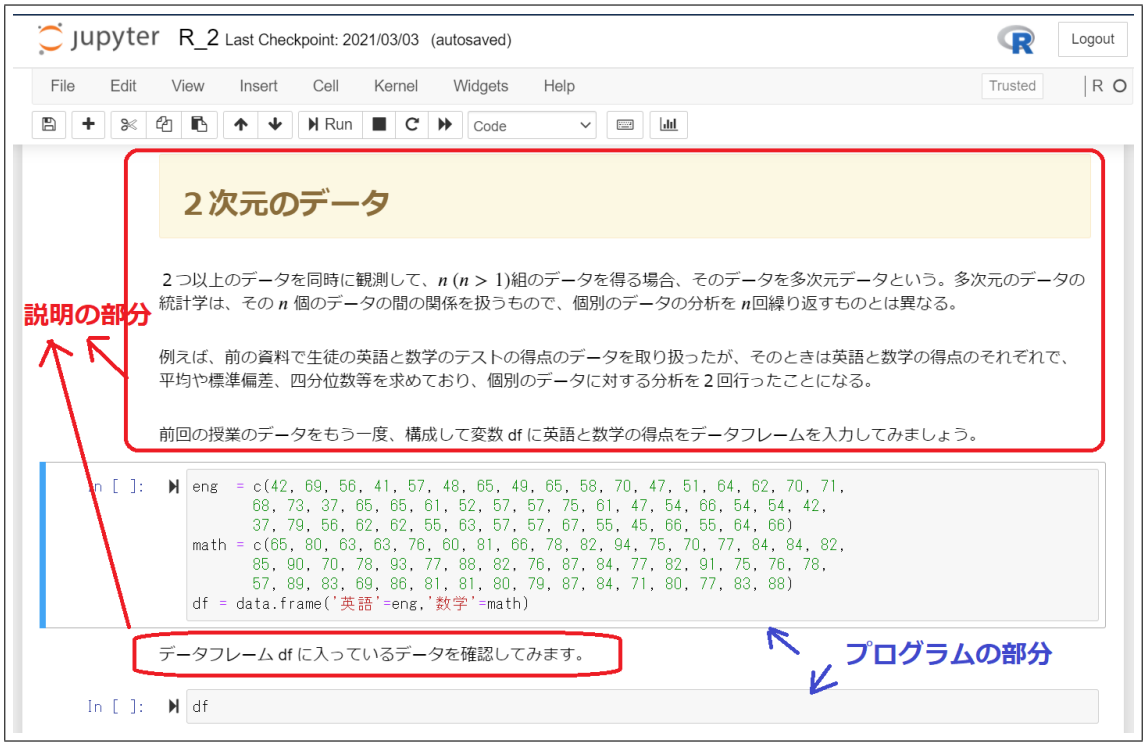


図 1： Jupyter Notebook の画面の例

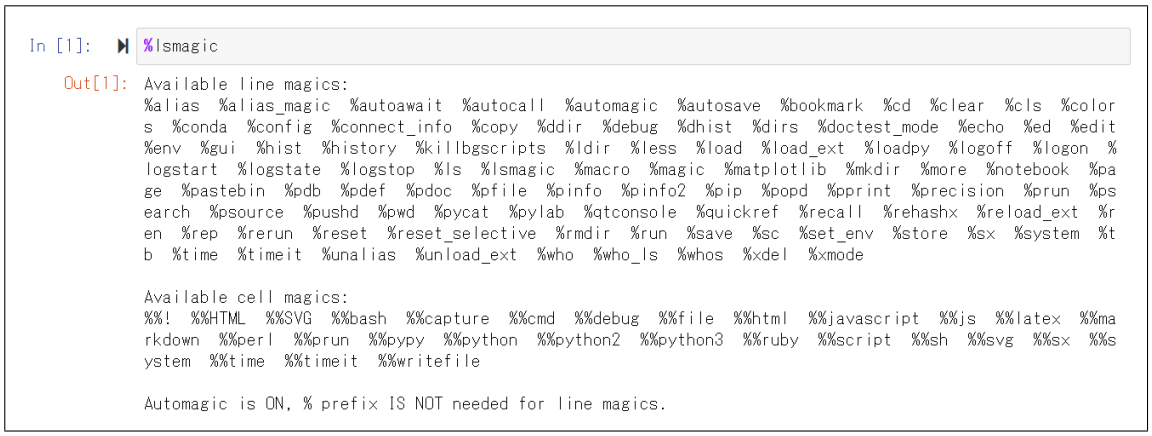


図 2：`%lsmagic` で利用可能な magic コマンドの一覧を調べる

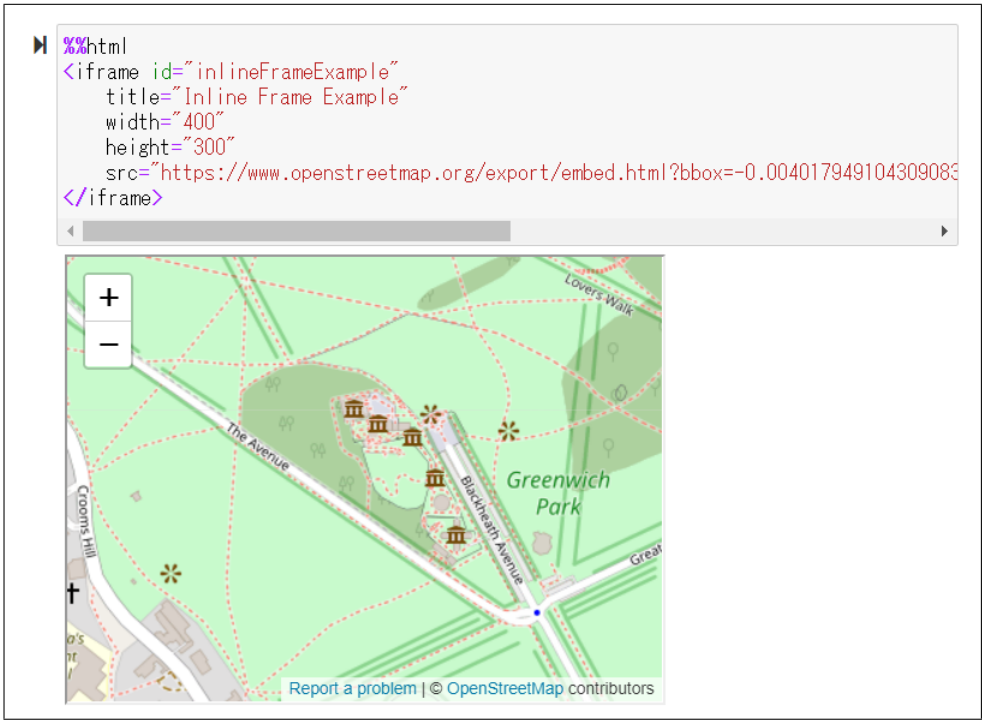


図 3：外部サイトの地図を Jupyter Notebook に埋め込む

`%%html` をセルの始めに記述すると、そのセルの中の命令はベースとなっているプログラミング言語の命令でなく、HTML の命令であると解釈される。よって、この magic コマンドを用いることで Jupyter Notebook の画面を HTML のタグを用いて操作したり、JavaScript の命令を差し込んだりすることが可能となる。

3　教材作成例

ここではマジックコマンド `%%html` を用いて Jupyter Notebook 上に作成した教材をいくつか示す。

3.1　Jupyter Notebook への外部の Web ページの埋め込み

`%%html` と HTML のタグ iframe を用いて、Jupyter Notebook に外部の Web ページを埋め込むことが可能である。それらを用いた教材例を示す。

3.1.1　教材の作成例

図 3 は iframe のタグを用いて、Jupyter Notebook に外部サイトの地図を埋め込んだものである。Python の命令を入力するセルに入力されている命令は下の通りである。

```
%%html
<iframe id="inlineFrameExample"
    title="Inline Frame Example"
    width="400"
    height="300"
    src="https://www.openstreetmap.org/export/embed.html?bbox=-0.004017949
    104309083%2C51.47612752641776%2C0.000305771827697753 96%2C51.4785698618
    98606&layer=mapnik">
</iframe>
```

1 行目にマジックコマンド %%html があるので、セルの中の命令は HTML として解釈される。2 行目以降は HTML の iframe タグを用いて外部のサイトを読み込んでいる。

次に図 4 に、動的幾何ソフトウェア Cinderella で作成した図を Jupyter Notebook に埋め込んだ例を示す。 Cinderella は動的幾何ソフトの 1 つであるが、作成した図を HTML

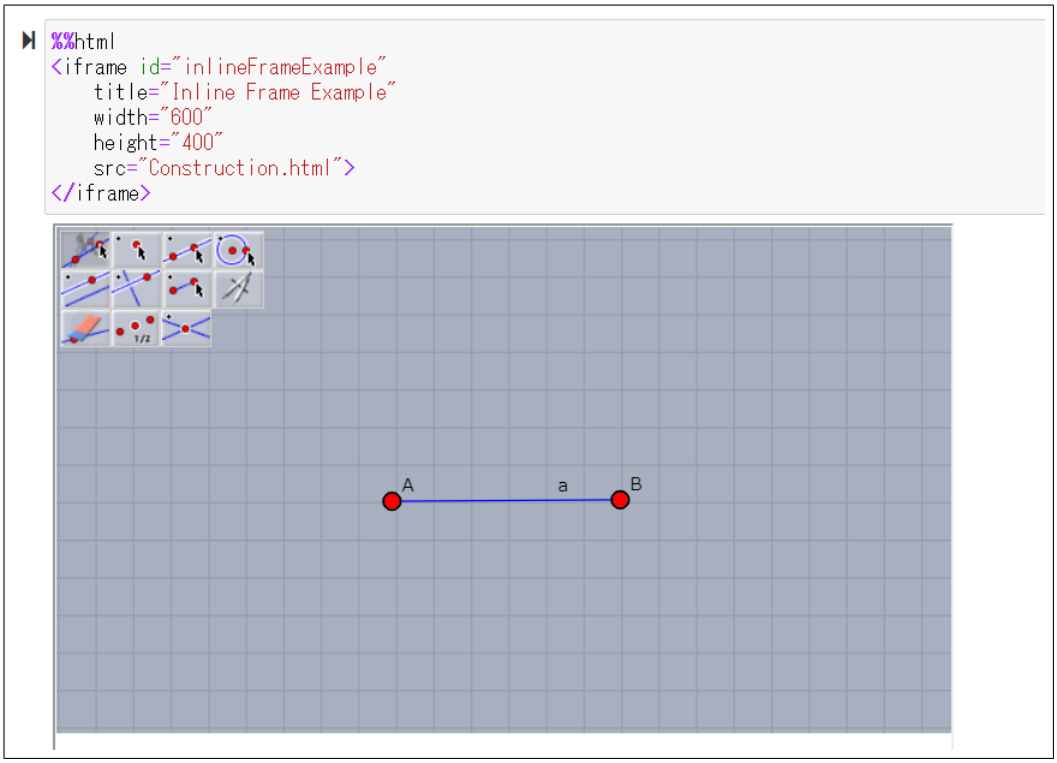


図 4：Cinderella で作成した図を Jupyter Notebook に埋め込む

ファイルに変換する事が可能である (HTML ファイルに変換した後も点や線を動かす事が可能)。この例では、Cinderella で作成した図を Contruction.html という名前の HTML に書き出して、それを Jupyter Notebook に読み込んでいる。

図 4 のセルに入力されている命令は下の通りである。

```
%%html
<iframe id="inlineFrameExample"
    title="Inline Frame Example"
    width="600"
    height="400"
    src="Construction.html">
</iframe>
```

先程と同様に、1 行目はマジックコマンド %%html であり、2 行目以降で HTML の iframe タグを用いて、Cinderella で作成したファイル Contruction.html を読み込んでいる。

3.1.2 本教材の意義・応用範囲

Jupyter Notebook へ埋め込んだ Web ページも通常の Web ページ同様にクリックやドラッグなどの操作が可能なので、静止画像を埋め込むのに比べて教材としての価値が高い。例えば、図 4 では点 A, B を動かしたり、左上のアイコンを用いて点、線や円を図に追加したりすることが可能である。

こういった外部の Web ページの埋め込みは通常のホームページでも可能であるが、Jupyter Notebook の機能と組み合わせることによってより使いやすい教材にできると考えられる。

3.2 Qull, jExcel の Jupyter Notebook への埋め込み

マジックコマンド %%html を用いると Jupyter Notebook に HTML の命令を埋め込むことができるが、これは script タグを用いることで JavaScript の命令を Jupyter Notebook に埋め込むことが可能であることを意味している。

JavaScript は近年、人気を高めつつあるプログラミング言語であり、様々なライブラリが開発されつつある。ここでは、JavaScript のライブラリである Quill と jExcel の Jupyter Notebook への埋め込みについて述べる。

Quill はブラウザ上で動作する WYSIWYG エディタの 1 つであり、JavaScript で書かれている。これを用いると、ブラウザ上に Word のような編集メニューを表示することが可能であり、Word で文書を作成するのと同様に、文章の入力や図の挿入などをブラウザ上で行うことが可能である。

また、jExcel はブラウザ上で動作する表計算ソフトウェアであり、こちらも JavaScript で書かれている。これを用いると、ブラウザ上に Excel と同様の表を表示させ、その表を編集することが可能になる。

3.2.1 教材の作成例

実際に Jupyter Notebook 上で Quill を動かした画面を図 5 に示す。図 5 のセルに入力されている命令は下の通りである。

```
%%html
<!-- Include stylesheet -->
<link href="https://cdn.quilljs.com/1.3.6/quill.snow.css"
 rel="stylesheet">
<!-- Create the editor container -->
<div id="editor">
  <p>Hello World!</p>
  <p>Some initial <strong>bold</strong> text</p>
  <p><br></p>
</div>
<script>
    requirejs.config({
        paths: {
            'quill': ['https://cdn.quilljs.com/1.3.6/quill'],
```

図 5：Quill の Jupyter Notebook への埋め込み

```
        },
    });
  require(['quill'], function(Quill) {
      new Quill('#editor', {theme: 'snow'});
  });
</script>
```

1 行目はマジックコマンド %%html で、2 行目以降は HTML であるがその中でタグ script を用いて、JavaScript の命令を埋め込んでいる。JavaScript の関数 Quill() が Quill を立ち上げる関数である。Jupyter Notebook 上で Quill を使うための注意点は、ライブラリ Quill の読み込みのために requirejs というモジュール管理用のフレームワークを使う必要があることである。

図 5 の画面下にサンプルの文章が「Hellow World!! ...」が入力されているが、ここに新しい文章を追加したり、その上の編集メニューを用いてこれらの文章を装飾したりすることが可能である。

次に、Jupyter Notebook 上で jExcel を動かした画面を図 6 に示す。この図のセルに入力されている命令は下の通りである。

```
%%html
<link rel="stylesheet" href="https://jsuites.net/v4/jsuites.css"
  type="text/css" />
<link rel="stylesheet"
href="https://bossanova.uk/jspreadsheet/v4/jexcel.css" type="text/css" />
<div id="my"></div>

<script>
requirejs.config({
  paths: {
    'jsuites': 'https://jsuites.net/v4/jsuites',
```

```
%%html
<link rel="stylesheet" href="https://jsuites.net/v4/jsuites.css" type="text/css" />
<link rel="stylesheet" href="https://bossanova.uk/jspreadsheet/v4/jexcel.css" type="text/css" />
<div id="my"></div>

<script>
requirejs.config({
  paths: {
    'jsuites': 'https://jsuites.net/v4/jsuites',
    'jexcel': 'https://bossanova.uk/jspreadsheet/v4/jexcel'
  },
  shim: {
    'jexcel': {
      deps: ['jsuites'],
      exports: '$.jexcel'
    },
  }
});
require(['jsuites','jexcel'], function($, jexcel) {
  data = [
    ['jExcel', 'Jquery spreadsheet, javascript spreadsheet, jquery', 181],
    ['Handsontable', 'Another nice javascript spreadsheet plugin', 9284],
    ['Datatables', 'DataTables is a table enhancing plug-in for the jQuery library.', 5164],
  ];
  jexcel(document.getElementById('my'), {
    data: data,
    colWidths: [300, 80, 100]
  });
});
</script>
```

	A	B	C
1	jExcel	Jquery spread	181
2	Handsontable	Another nice j	9284
3	Datatables	DataTables is	5164

図 6：jExcel の Jupyter Notebook への埋め込み

```
    'jexcel': 'https://bossanova.uk/jspreadsheet/v4/jexcel'
  },
  shim: {
    'jexcel': {
      deps: ['jsuites'],
      exports: '$.jexcel'
    },
  }
});
require(['jsuites','jexcel'], function($, jexcel) {
  data = [
    ['jExcel', 'Jquery spreadsheet, javaScript spreadsheet, jquery', 181],
    ['Handsontable', 'Another nice javaScript spreadsheet plugin', 9284],
    ['Datatables',
     'DataTables is a table enhancing plug-in for the jQuery library.',
     5164],
  ];
  jexcel(document.getElementById('my'), {
    data: data,
    colWidths: [300, 80, 100]
  });
});
</script>
```

Quill の場合と同様に、タグ script を用いて、JavaScript の命令を埋め込んでおり、JavaScript

の関数 jexcel() で jExcel を立ち上げている。その時に、引数の連想配列のキー `data` で jExcel 表のデータ、`colWidths` で表のサイズをそれぞれ指定している。

図 6 の画面下にサンプルの表のデータが入力されているが、ここの表のデータを変更したり (通常の表計算と同様に簡単な計算も行える)、表に新しい行を追加したりすることが可能である。

3.2.2 本教材の意義・応用範囲

Quill を用いて Word のような文書形成機能を Jupyter Notebook に埋め込むことで、学生のレポート作成の補助になると期待できる (正確には Jupyter Notebook では Markdown の記法を用いて文書形成が行えるが、多くの学生は Word による文書形成の方が慣れていると思われる)。

また、jExcel を用いて作成した表は表示だけでなく、編集も行えるため (表に直接データを入力したり、他のセルを用いた簡単な計算が可能である)、作成した教材をより使いやすくするために有用であると思われる。

3.3 日本語の分かち書きを行う教材

ここでは、Python のライブラリを用いて作成した日本語の分かち書きを行う教材を紹介する。

3.3.1 教材の概要

教材の全体像を図 7 に示す。図 7 からわかるようにこの教材は 2 つのセルからなる。上のセルでは分かち書きの計算を行う Python の関数を定義している。下のセルではマジックコマンド `%%html` を用いて、HTML によりテキストボックスやボタンの作成、並びに、処理を行う JavaScript の関数定義等を行っている。

3.3.2 教材の使い方

Jupyter Notebook は基本的には対話的に使うように設計されており、キーボードからコマンドを行い、そのコマンドによる計算結果がそのつど画面に出力されるようになっている。つまり、Jupyter Notebook を使うためにはベースとなっているプログラミング言語 (Python が多い) を理解し、その文法に従い、入力を構成する必要がある。よって、Jupyter Notebook は使い人を選ぶようになっており、このことが Jupyter Notebook の教材活用を制限している。

この問題を解決するために、本教材はプログラミングの知識を全く持たなくても使えるように設計されている。本教材の使い方は次の通りである。

(s1) 「(1) まず、このボタンをクリックする」のボタンをクリックする。そうすると、`IPython.notebook.execute_all_cells()` が実行され、全てのセルの内容が実行される。また、図 8 のように画面からセルの内容が見えなくなる (これは後に説明するように関数 `code_toggle()` が実行された結果である)。

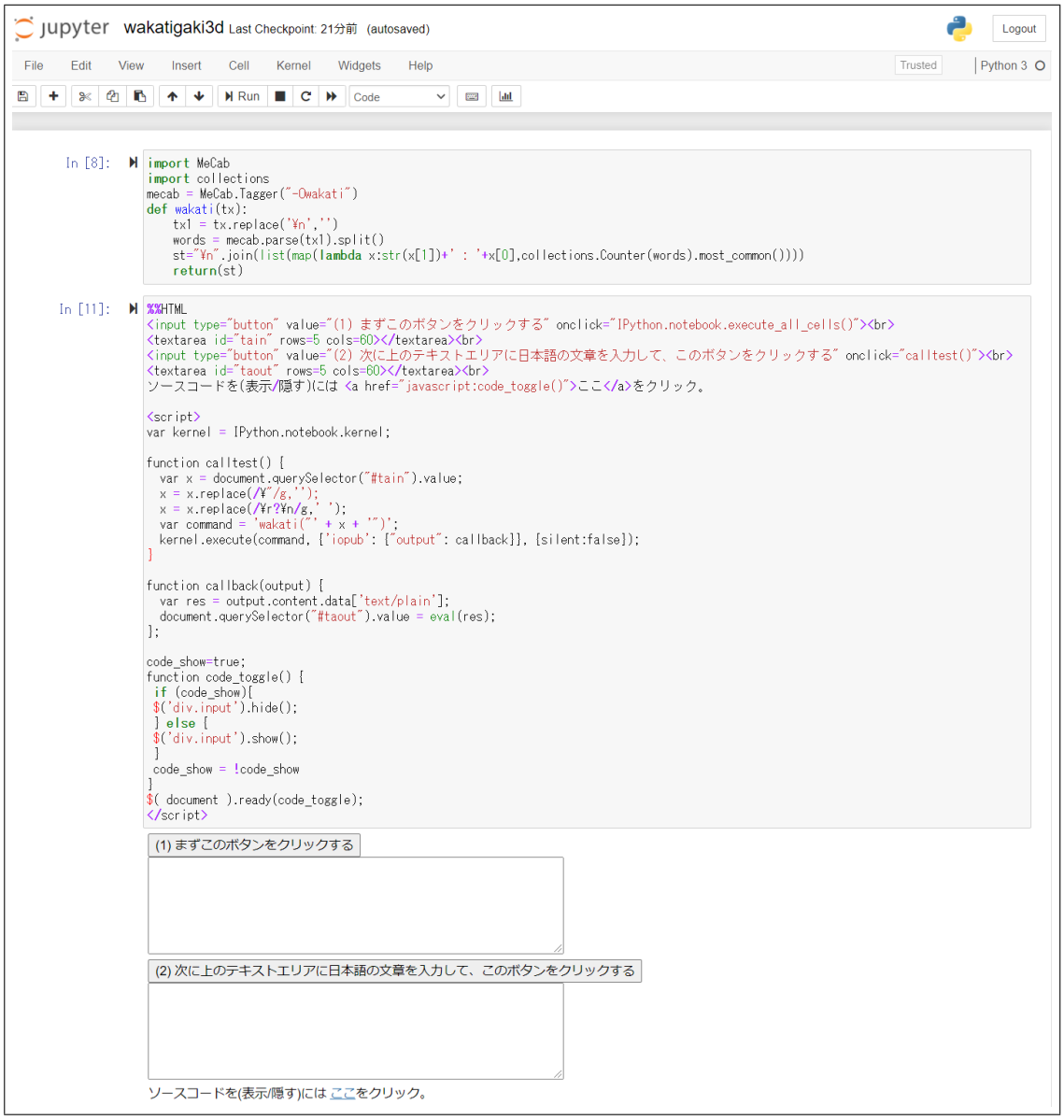


図７：分かち書きを行うアプリの全体像

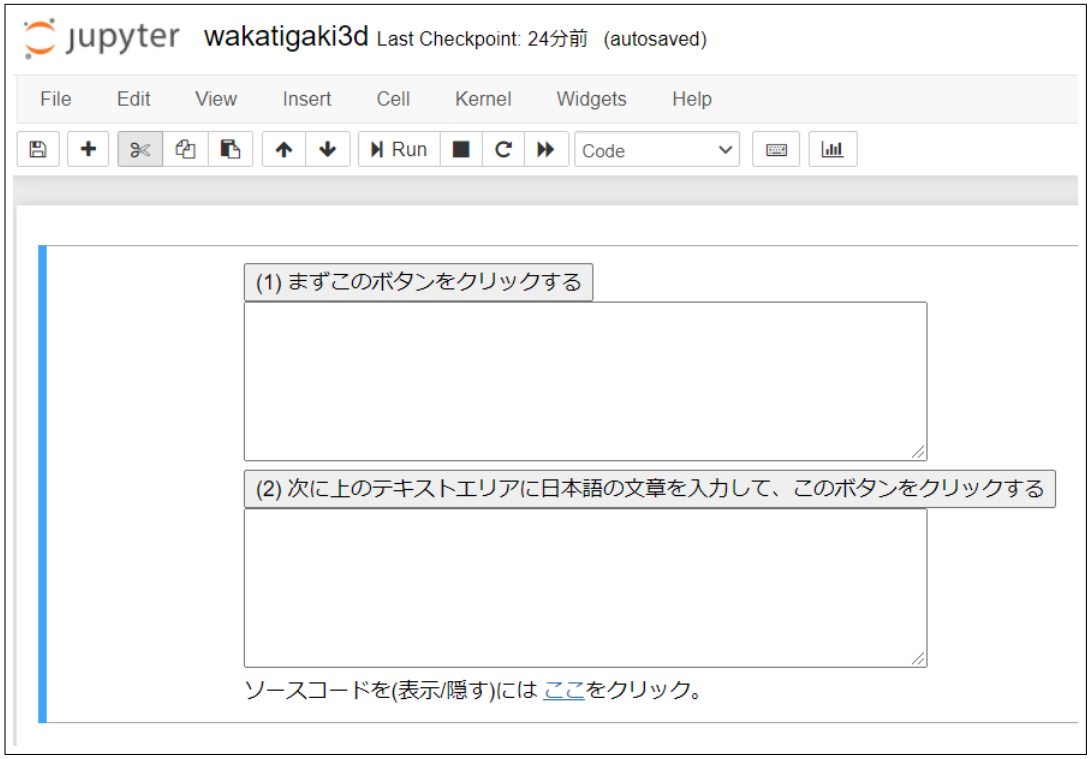


図 8：分かち書きを行うアプリの実行画面 (1)

(s2) 2 つあるテキストボックスの上のものに適当な日本語の文章を入力する。

(s3) 「(2) 次に上のテキストエリアに日本語の文章を入力して、このボタンをクリックする」をクリックする。

(s4) 2 つあるテキストボックスの下のものに入力された日本語を分かち書きした結果が表示される (図 9 を参照)。

3.3.3 本教材の内容の解説

ここでは、図 7 のセルの内容、すなわちこの教材のプログラムについて解説する。まず、図 7 にはセルが 2 つあるが、上のセルの中身は Python のコードになっており、次のようになっている。

```
import MeCab
import collections
mecab = MeCab.Tagger("-Owakati")
def wakati(tx):
    tx1 = tx.replace('\n','')
    words = mecab.parse(tx1).split()
    st="\n".join(list(map(lambda x:str(x[1])+' : '+x[0],
         collections.Counter(words).most_common())))
    return(st)
```

図 9：分かち書きを行うアプリの実行画面 (2)

ここでは MeCab というパッケージを用いて、分かち書きを行う関数 `wakati(tx)` を定義している。この関数は文字列の引数 `tx` を入力として、それを分かち書きした結果を文字列にして返している。

次に 2 番目のセルの内容を解説する。セルの内容は下のようになっている。

```
%%HTML
<input type="button" value="(1) まずこのボタンをクリックする"
  onclick="IPython.notebook.execute_cells_below()"><br>
<textarea id="tain" rows=10 cols=60></textarea><br>
<input type="button" value="(2) 次に上のテキストエリアに日本語の文章を
入力して、このボタンをクリックする" onclick="calltest()"><br>
<textarea id="taout" rows=10 cols=60></textarea><br>
ソースコードを(表示/隠す)には <a href="javascript:code_toggle()">
 ここ</a>をクリック。

<script>
var kernel = IPython.notebook.kernel;

function calltest() {
  var x = document.querySelector("#tain").value;
  x = x.replace(/\"/g,'');
  x = x.replace(/\r?\n/g,' ');
  var command = 'wakati("' + x + '")';
  kernel.execute(command,{'iopub':{"output": callback}},{silent:false});
}

function callback(output) {
```

```
  var res = output.content.data['text/plain'];
  document.querySelector("#taout").value = eval(res);
};

code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
```

まず、HTML の input タグでボタンを、textarea タグでテキストエリアをそれぞれ生成する。また、図 8,9 の画面の下にある「ソースコードを (表示/隠す) にはここをクリック。」の説明文を入れ、「ここ」をクリックすると JavaScript の関数 `code_toogle()` が呼び出されるように設定する。

次に script タグで JavaScript のプログラムを挿入する。

```
var kernel = IPython.notebook.kernel;
```

は、JavaScript から Jupyter Notebook のセルにアクセスするための設定である。また、ここで JavaScript の関数 `calltest()`, `callback()`, `code_toggle()` の定義を行う。それぞれの関数の内容は以下の通りである。

> `calltest()` : 分かち書きを行う関数で、上のテキストボックス (id が tain のテキストボックス) 日本語の文章を受け取り、変数 `x` に代入した後、「”」を削除し、改行コードを空白「 」で置き換える。その後、Jupyter Notebook の命令 `kernel.execute()` で Python の命令 `wakati()` を引数を `x` にして呼び出し、計算結果を下のテキストボックスに出力する。
>
> `callback()` : Python からの返り値を下のテキストボックス (id が taout のテキストボックス) に書き出す関数である。この関数を `kernel.execute()` に引数として与えることで Python での計算結果をテキストボックスに出力する。
>
> `code_toggle()` : Jupyter Notebook のセルの中身の表示・非表示を行う関数である。呼び出されるごとに表示・非表示の切り替えを行う。

教材の画面にある「(2) 次に上のテキストエリアに日本語の文章を入力して、このボタンをクリックする」のボタンをクリックすると `calltest()` が呼び出され、上のテキストエリアにある文章を分かち書きしたものが下のテキストエリアに出力される。

3.3.4 本教材の意義・応用範囲

この Web アプリは Python のパッケージ “MeCab” を用いて日本語の文章の分かち書きを行っているが、テキストボックスに日本語を入力してクリックするだけですぐに使え、

図 10：分かち書きを行い、その結果を PDF ファイルに出力する教材

Python の命令を入力する必要がないため、授業教材として使いやすい。Python の関数 `wakati()` を置き換えるだけで分かち書きだけでなく様々な Python のパッケージを用いた Web アプリを作成することが可能で、応用範囲に広い例だと考えられる。

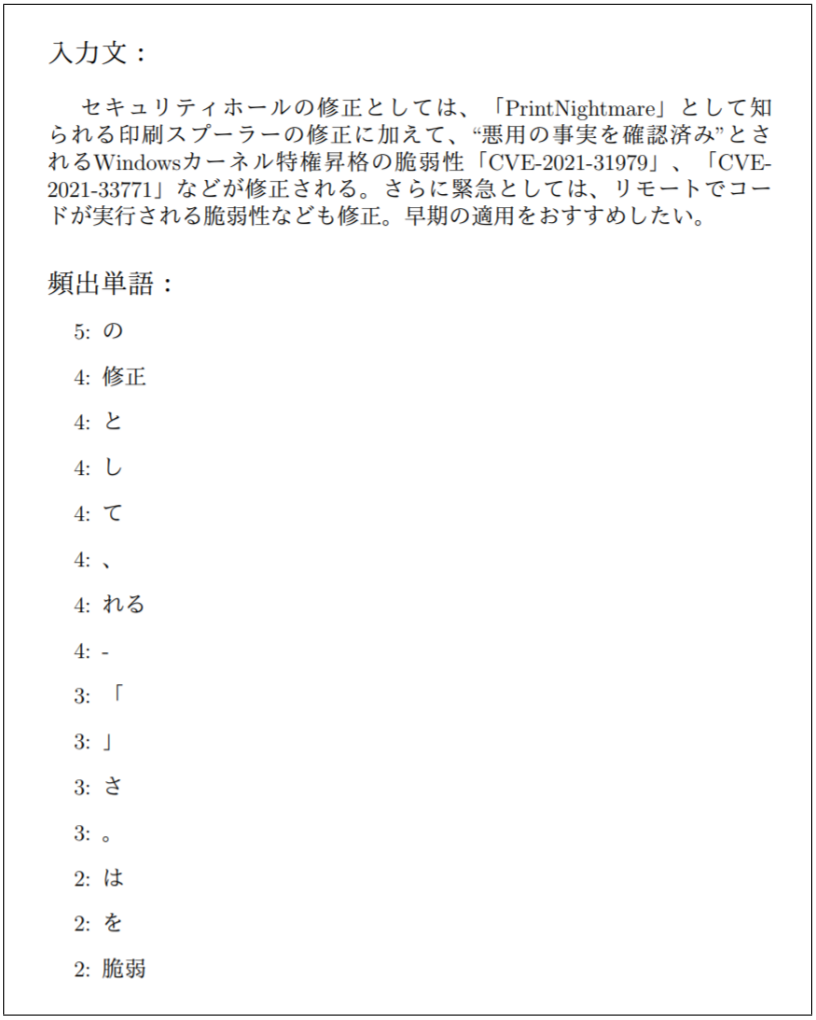
3.4 Jupyter Notebook から外部の Web API へのアクセス

HTML のコマンドを用いて、Jupyter Notebook から外部の Web API へアクセスし、そのサービスを活用することもできる。現在、様々な無料の外部 Web サービスが存在しており、例えば、LaTeX.Online というサイト (参考文献 [4]) では、Latex のソースファイルから PDF を作成する Web API を提供している。これを用いると TeX をコンパイルするためのソフトウェアをパソコンにインストールしなくても TeX のソースファイルをコンパイルし、PDF ファイルを作成することが可能となる。Overleaf (参考文献 [5]) も同様のオンライン上の TeX の環境を提供しているが、こちらはユーザーがブラウザでアクセスして TeX を使う形を取っており、Web API の形を取っていない。

3.4.1 教材の作成例

ここでは、上で述べた LaTeX.Online の Web API を活用した教材例を紹介する。教材の画面を図 10 に示す。この教材は上で既に紹介した分かち書きを行うものに、その結果を PDF ファイルで出力する機能を付け加えたものであり、次のようにして使う。

(s1) 「(1) まず、このボタンをクリックする」のボタンをクリックする。


入力文：

セキュリティホールの修正としては、「PrintNightmare」として知られる印刷スプーラーの修正に加えて、“悪用の事実を確認済み”とされるWindowsカーネル特権昇格の脆弱性「CVE-2021-31979」、「CVE-2021-33771」などが修正される。さらに緊急としては、リモートでコードが実行される脆弱性なども修正。早期の適用をおすすめしたい。

頻出単語：

5: の
4: 修正
4: と
4: し
4: て
4: 、
4: れる
4: -
3: 「
3: 」
3: さ
3: 。
2: は
2: を
2: 脆弱

図 11：出力された PDF ファイルの例

(s2) 3 つあるテキストボックスの一番上のものに適当な日本語の文章を入力する。

(s3) 「(2) 次に上のテキストエリアに日本語の文章を入力して、このボタンをクリックする」のボタンをクリックする。

(s4) 分かち書きの結果が、3 つあるテキストボックスの真ん中のものに日本語の文章で、一番下にものに TeX のソースファイルの形で出力される。同時に、自動的に外部サイト LaTex.Online へのアクセスが行われ、そこで TeX のソースファイルのコンパイルが行われ、結果の PDF ファイルが出力される。出力された PDF ファイルの例を図 11 に示す。

3.4.2 本教材の内容の解説

上で述べたように、この教材は **3.3** 節で解説した「分かち書きを行う教材」に、PDF ファイルで結果を出力する機能を付け加えたものである。そのために追加した部分を中心

に、本教材の内容の解説する。

まず、図 10 からわかるように、テキストエリアが 1 つ増えて 3 つになっている。上の使い方の説明の (s2) で説明したように、追加された最後のテキストエリアには、PDF ファイルの元となる TeX のソースファイルが出力される。

次に実際に処理を行うプログラムを、**3.3** 節の「分かち書きを行う教材」との違いを中心に解説していく。「(2) 次に上のテキストエリアに日本語の文章を入力して、このボタンをクリックする」のボタンをクリックすると呼び出される `calltest()` は次のように修正されている。

```
function calltest() {
  var x = document.querySelector("#tain").value;
  x = x.replace(/\"/g,'');
  x = x.replace(/\r?\n/g,' ');
  var command = 'wakati("' + x + '")';
  kernel.execute(command,{'iopub':{"output": callback}},{silent:false});
  var command2 = 'wakati2("' + x + '")';
  kernel.execute(command2,{'iopub':{"output": callback2}},{silent:false});
}
```

上からわかるように、**3.3** 節の「分かち書きを行う教材」での `calltest()` と比較すると、Python の関数 `wakati2(x)` の呼び出しが新たに追加されている。この `wakati2(x)` は、分かち書きの結果 `x` を引数として与えられ、TeX のソースファイルを返す関数であるが、その関数定義は次のようになっている。

```
def wakati2(tx):
    tx1 = tx.replace('\n','')
    words = mecab.parse(tx1).split()
    st0 = '''
% encoded in UTF-8
\\documentclass[a4j,12pt]{article}
\\usepackage{CJKutf8}
\\begin{document}

\\begin{CJK}{UTF8}{ipxm}
\\noindent
{\\large 入力文：}

\\vspace*{0.5cm}
'''
    st0 = st0 + tx
    st0 = st0 + '''
\\end{CJK}

\\vspace*{0.8cm}

\\begin{CJK}{UTF8}{ipxm}
\\noindent
{\\large 頻出単語：}
\\end{CJK}
\\begin{itemize}
'''
  st="\n".join(list(map(lambda x:'\\item['+str(x[1])+':]
      \\begin{CJK}{UTF8}{ipxm}'+x[0]+'\\end{CJK}',
     collections.Counter(words).most_common()[:15])))
  st0 = st0 + st + '''
```

```
\\end{itemize}
\\end{document}
'''
    return(st0)
```

次の JavaScript の関数 `callback2(output)` は、この `wakati2(x)` で生成した TeX のソースファイルを受け取り、最後のテキストエリアに書き込むと同時に LaTeX.Online にアクセスして TeX のソースファイルをコンパイルした PDF ファイルを受け取り画面に表示する。

```
function callback2(output) {
  var res = output.content.data['text/plain'];
  document.querySelector("#tex").value = eval(res);
  texcompile();
};

function texcompile() {
  var x = encodeURI(document.querySelector("#tex").value);
  var url = 'https://latexonline.cc/compile?text=';
  window.open(url+x, '_blank');
}
```

`callback2(output)` で別の JavaScript の関数 `texcompile()` を呼び出しているのでその定義も示している。なお、関数 `texcompile()` では、JavaScript の関数 `window.open()` を用いて、外部のサイトにアクセスしている。

3.4.3 本教材の意義・応用範囲

この教材では日本語の分かち書きを行っているが、分かち書きを行う python の関数 `wakati(tx)` と TeX のソースコードを生成する関数 `wakati2(tx)` を置き換えれば、色々な計算結果を PDF ファイルで出力する教材を作成することができる。python には様々なライブラリが準備されているので、応用範囲は広いと思われる。

4 外部の Jupyter Notebook サーバーの活用

これまで、Jupyter Notebook を活用した例を示してきたが、Jupyter Notebook は Web ベースのアプリであるので Jupyter Notebook を使うためには Jupyter Notebook のサーバーが必要である。Windows などのパソコンならば Anaconda (参照文献 [6]) などをインストールすることによりパソコン自体をサーバーとして用いることも可能であるが、この場合は Anaconda などのソフトウェアのインストールが必要となる上、自分以外の人がその Jupyter Notebook にアクセスすることはできない。つまり、図 12 に示しているように、教員が自分のパソコンに Anaconda をインストールして自分のパソコンで Jupyter Notebook が使えるようにしても、生徒からのパソコンからは教員の Jupyter Notebook にはアクセス出来ないため教材としては使いにくい。

そこで、外部の Jupyter Notebook のサーバーを活用することを考える。Google のアカウントを保持していれば、Google の提供するサービスである Google Colaboratory (参考

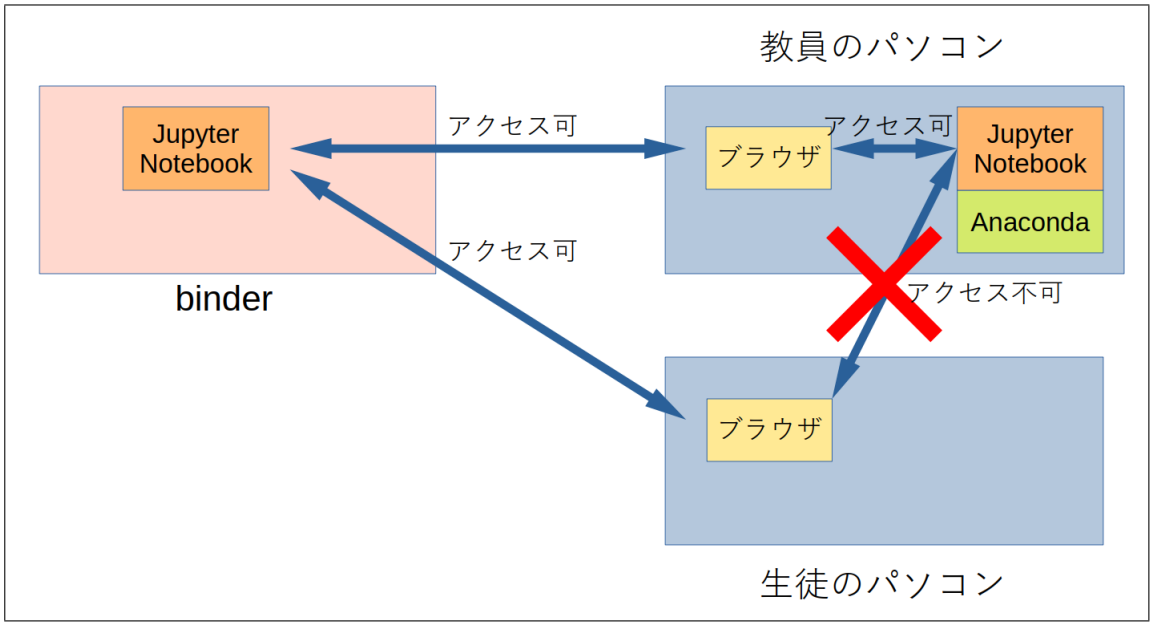


図 12： Jupyter Notebook へのアクセスの可否

文献 [7]) を用いることも可能であるが、ここでは binder(参考文献 [8]) という Web サービスに注目し、これを活用した教材運用を考える。

4.1 binder とは

binder は無料で使える Web サービスの 1 つである。GitHub の repository に Jupyter Notebook のファイルを置いておくと、それを元にした Jupyter Notebook の環境を構築して、誰でも使えるようにしてくれる。図 12 にあるように、binder で作成した Jupyter Notebook の環境は URL さえ知っていれば誰でもアクセスが可能である。つまり、Jupyter Notebook の教材を GitHub の repository に置いておけば、自分でサーバーを用意しなくても、生徒がブラウザからその教材を使えるようになる。

4.2 活用例

実際に分かち書きを行う Jupyter Notebook を binder 上で使えるように設定し、ブラウザでこの URL にアクセスして分かち書きを行った画面を図 13 に示す。

5 まとめと今後の課題

近年、Jupyter Notebook の人気が高まりつつあるが、本論文では Jupyter Notebook 上での教材作成について述べた。

Jupyter Notebook はブラウザ上で動作し、マジックコマンドを用いることで、HTML の命令を埋め込むことが可能である。そこで、JavaScript 等を活用することで、Jupyter

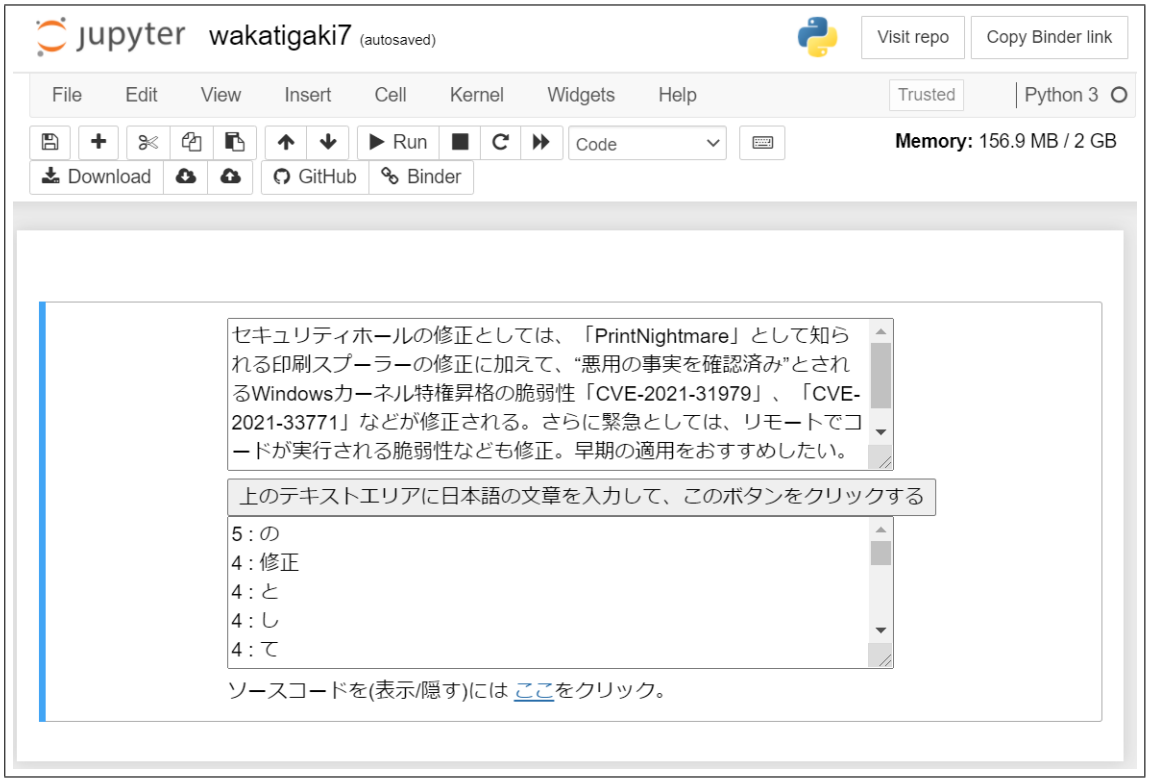


図 13：binder 上の Jupyter Notebook で動かした分かち書きの教材

Notebook 上で次のような方法での教材作成が可能であることを示し、教材の作成例を与えた。

- Jupyter Notebook への外部の Web ページの埋め込む。
- Qull, jExcel などの JavaScript のライブラリを Jupyter Notebook へ埋め込む。
- HTML の命令を埋め込むことで、ボタンをクリックするだけで動作する教材を作成する (その例として、日本語の分かち書きを行う教材を示した)。
- 外部の Web API にアクセスし、そのサービスを活用する。(その例として、日本語の分かち書きを行う教材に、分かち書きの結果を PDF ファイルとして出力する機能を追加する例を示した)

また、Jupyter Notebook を生徒に使わせるためには、生徒のパソコン１台１台に Anaconda をインストールするか、Jupyter Notebook のサーバーを準備する必要があるが、無料のサービスである binder を使うことでこの問題をクリアできることを示した。

今後は、参考文献 [2], [3] で行っているような E-Learning システムなどを Jupyter Notebook 上で実現することを課題として研究を進めていきたい。

参考文献

[1] 北本卓也, Web 上の教材作成について, 2019 年度 RIMS 共同研究 "Computer Algebra - Theory and its Applications", 京都, 2019 年 12 月.

[2] T. Kitamoto, M. Kaneko, S. Takato, E-learning system with Computer Algebra based on JavaScript programming language, Proc. of ATCM 2018, pp. 123-133, Yogyakarta, 2018.

[3] 数学教員のための HTML 教材, URL: https://sattch.github.io/math-lab/ (2021 年 4 月 30 日閲覧)

[4] LaTeX.Online Official Homepage, https://latexonline.cc/ (2021 年 4 月 30 日閲覧)

[5] Overleaf Official Homepage, https://ja.overleaf.com/ (2021 年 4 月 30 日閲覧)

[6] Anaconda Official Homepage, https://www.anaconda.com/ (2021 年 4 月 30 日閲覧)

[7] Google Colaboratory Official Homepage, https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja (2021 年 4 月 30 日閲覧)

[8] binder Official Homepage, https://mybinder.org/ (2021 年 4 月 30 日閲覧)