МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет  
Информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Кафедра информационных систем

Лабораторная работа №5

Разработка плагинов

Выполнил студент   
группы № М3309:

Ефремов Иван

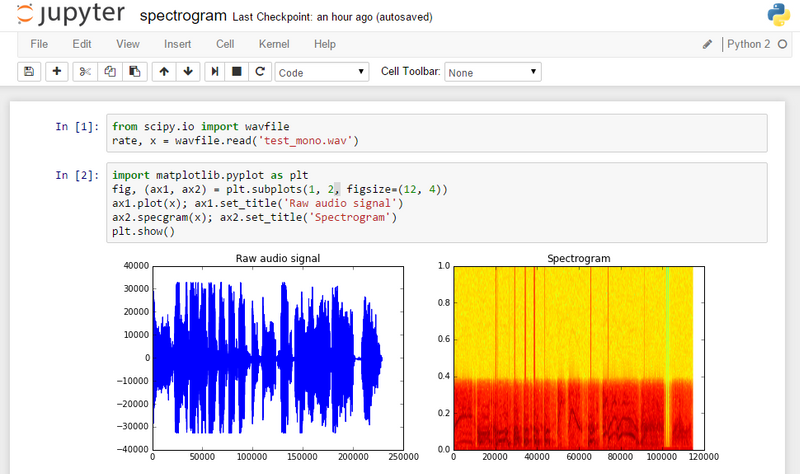
САНКТ-ПЕТЕРБУРГ  
2018

Jupyter Notebook

**Jupyter Notebook** - популярнейшая бесплатная интерактивная оболочка для языка программирования **Python**, позволяющая объединить код, текст и диаграммы, и распространять их для других пользователей.

Раньше она называлась **IPython Notebook**, но название сменили, чтобы подчеркнуть совместимость не только с **Python**, но и другими языками программирования.

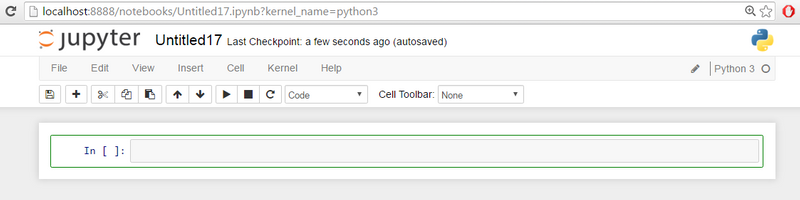
Это крайне удобный инструмент для проведения экспериментов (как чисто математических, так и связанных с обработкой данных от приборов) и оформления результатов в виде профессиональных научных статей. А вообще человечество нашло ему бесчисленные применения - от дизайна электронных схем до обработки астрономических фотографий. Особенно в последние годы Jupyter Notebook обрёл популярность среди аналитиков (дата сайнтистов).



Jupyter поддерживает множество языков программирования и может быть легко запущен на любом сервере, необходим только доступ по ssh или http. К тому же это свободное ПО.

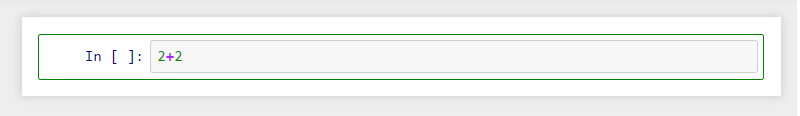
Работа в Jupyter Notebook

Запустим Jupyter Notebook. В браузере открывается новая вкладка. Это и есть наша новая чистая тетрадь. Вверху расположены разнообразные элементы управления, а зеленым выделена единственная пока ячейка.



Да, тетради здесь состоят из ячеек. Это очень удобно. Настолько удобно, что люди, как правило, перестают пользоваться текстовыми редакторами типа Word и пишут в этих тетрадках не только программы, но и всё подряд - от книг до научных статей.

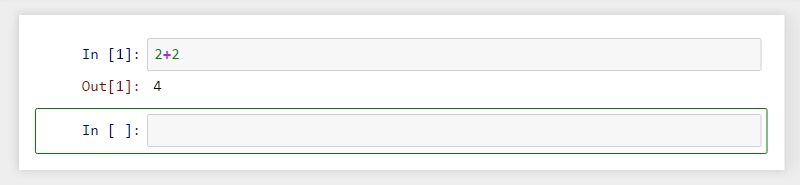
Давайте что-нибудь напишем в нашей ячейке. Например, **2+2**



Теперь для запуска кода на исполнение нажмите вот такую кнопку:

http://openfablab.ru/play/default/picture?img=366

И получите результат:

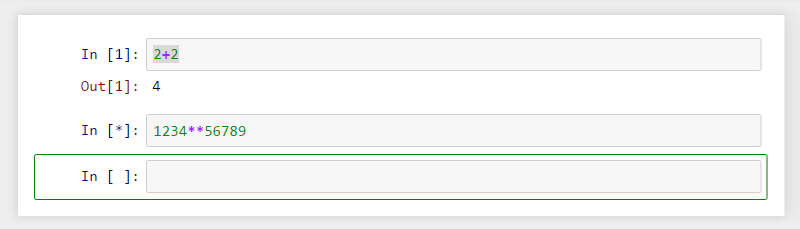


Внизу нашей ячейки появилось поле с результатом. Кроме того, вы видите метки **In[1]** и **Out[1]**. Эти метки, как мы ещё увидим, позволяют использовать содержимое ячеек в дальнейших вычислениях.

А ещё возникла новая ячейка - для новых задач. Можно задать здесь что-то новое, а можно отредактировать программный код в старой ячейке и запустить ее заново.

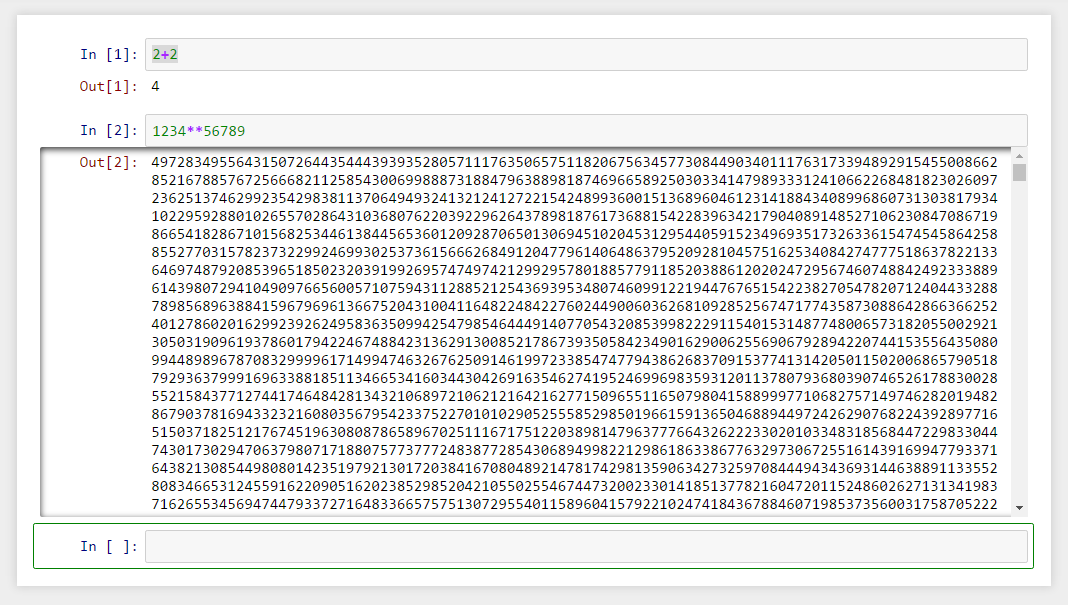
Ну, например, давайте узнаем чему равно **123456789**. В языке **Python** умножение обозначается знаком звездочки: \*, а возведение в степень - двойной звездочкой: \*\*

После того, как я набрал это выражение и запустил его, опять возникла новая ячейка. Но вместо ответа слева от задания появилась метка со звездочкой: **In[\*]**



Это значит, что программа находится в процессе вычислений. Ну это логично - выражение мы задали не самое простое.

А через несколько секунд появился результат. Содержащий сотни тысяч цифр! Они, конечно, не уместились на экран и поэтому справа видна полоса прокрутки для пролистывания ответа.



Расширения

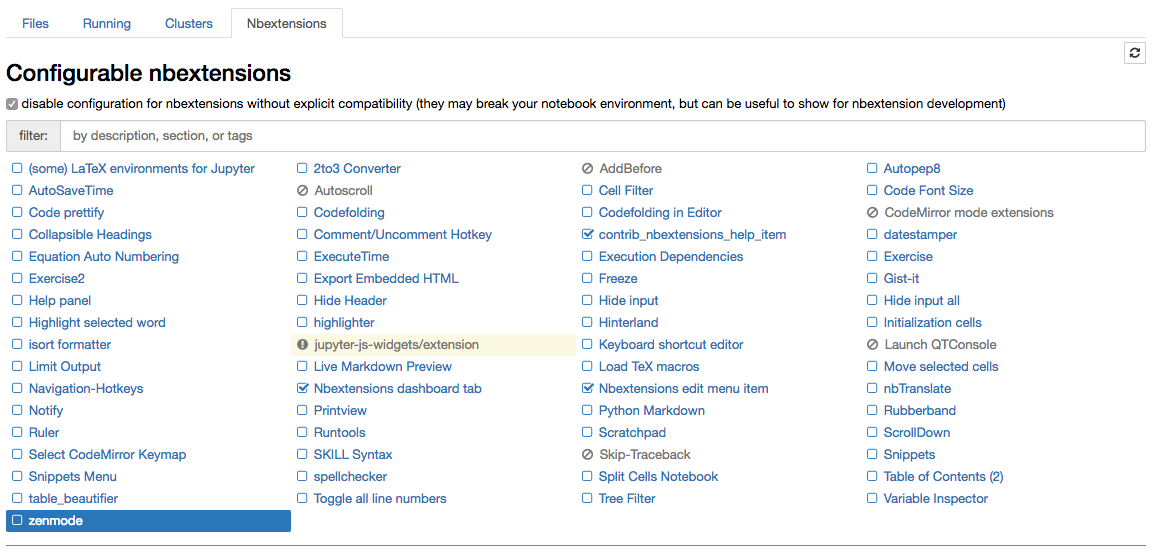
Как известно, Jupyter Notebook не блещет функциональностью. По сути, кроме ячеек там ничего-то и нет. Но выход, конечно, есть. Есть множество способов «прокачать» свой Jupyter. Один из них – это установка и/или создание расширений.

Существует библиотека *jupyter\_contrib\_nbextensions* уже с готовыми расширениями. Чтобы ими было удобно управлять придумали специальный плагин, дающий интерфейс для расширений, который называется *jupyter\_nbextensions\_configurator.*

Установим эти библиотеки

pip install jupyter\_nbextensions\_configurator jupyter\_contrib\_nbextensions   
jupyter contrib nbextension install --user   
jupyter nbextensions\_configurator enable --user

Перезагружаем jupyter



Теперь мы можем одним кликом включить или отключать нужное нам расширение, правда тетрадку придется перезагружать. Также данный плагин поддерживает создание своих расширений.

Структура расширения

Каждое расширение минимум состоит из 2 файлов:

1. description.yaml : Файл конфигурации
2. main.js : Javascript код для самого расширения

Можно добавить ещё один файл – readmy.md – описание расширения с помощью markdown. Поместим эти файлы в папку, которую назавём lighter – имя расширения, которое мы собираемся создать. В свою очередь lighter должна лежать в папке *nbextensions*, которая является поддиректорией *jupyter\_contrib\_extensions (jupyter\_contrib\_extensions –* установленная раннее через pip библиотека). Таким образом, полный путь до расширения в моём случае - C:\Users\Efrem\Anaconda3\Lib\site-packages\jupyter\_contrib\_nbextensions\nbextensions\lighter.

Структура расширения:

nbextensions/   
 lighter/  
 - description.yaml  
 - main.js  
 - README.md

При разработке расширения, чтобы изменения пришли в силу, нужно запустить команду *jupyter contrib nbextensions install –user* и перезапустить «тетрадку».

Давайте теперь пройдёмся по каждому файлу.

**Yaml файл – lighter.yaml**

**YAML** — «дружественный» формат сериализации данных, концептуально близкий к языкам разметки. YAML описывает расширение для конфигуратора для правильного отображения в списке всех плагинов. В файл таже записываются все параметры, которые потом можно изменять вручную.

Мой YAML файл:

Type: IPython Notebook Extension

Name: Lighter

Description: Higlighting cells

Link: readme.md

Icon: icon.png

Main: lighter.js

Compatibility: 4.x, 5.x

Parameters:

- name: Lighter.colors.successful\_exec

description: |

Color to use for highlighting successful finish after execution

default: '#00BB00'

input\_type: color

- name: Lighter.colors.error\_exec

description: |

Color to use for highlighting fail finish after execution

default: '#cc0000'

input\_type: color

- name: Lighter.colors.execution

description: |

Color to use for highlighting during execution

default: '#42A5F5'

input\_type: color

- name: Lighter.colors.waiting

description: |

Color to use for highlighting cell, which is waiting for execution

default: '#FFFF66'

input\_type: color

- name: toogle\_plugin\_hotkey

description: Hotkey combination to which to bind plugin-toggling

input\_type: hotkey

default: Alt-N

- name: button\_enable

description: Enable the button to toogle plugin

input\_type: checkbox

default: true

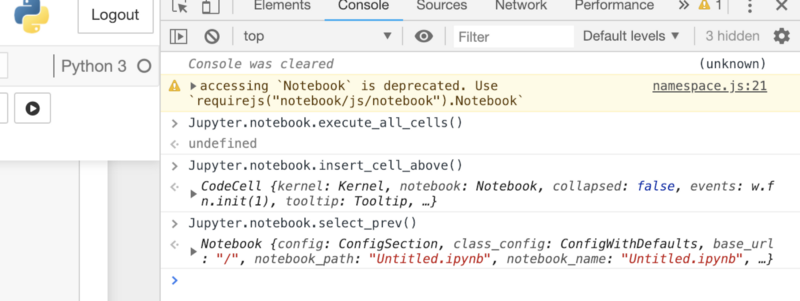
Основные параметры:

* **Name** – имя расширения
* **Description** – краткое описание расширения
* **Link** – ссылка на более подробное описание, в данном случае это markdown файл
* **Icon** – изображение к плагину
* **Main** – имя основного файла в директории, в данном случае это lighter.js
* **Compatibility** – совместимость с версией Jupyter notebook
* **Parameters** – параметры расширения, которые можно изменять вручную
  + **Lighter.colors.successful\_exec** – цвет ячейки после успешного выполнения
  + **Lighter.colors.error\_exec** – цвет ячейки после выполнения с ошибкой.
  + **Lighter.colors.execution** – цвет ячейки при выполнении
  + **Lighter.colors.waiting** – цвет ячейки, которая находится в очереди на выполнение
  + **toogle\_plugin\_hotkey** – горячая клавиша на включение/выключение плагина
  + **button\_enable** – добавление кнопки в меню кнопок

**Javascript файл – lighter.js**

Это сердце расширения, где расположена вся его логика. Jupyter Notebook запускается в браузере, а это значит, что само расширение должно быть написано на Javascript, язык веб-разработки.

Документации практически нет, поэтому сложно понять какие команды использовать. Один из самых проверенных и надёжных способов – запускать команды в консоле браузера и находить нужные нам. Chrome developer tools -cntrl + shift + i.

****

Открываем консоль и ищем команды, начинающиеся с Jupyter.notebook. Сразу же в тетрадке можно видеть результат выполнения.

Мой Javascript файл:

define([

'require',

'jquery',

'moment',

'base/js/namespace',

'base/js/events',

'notebook/js/codecell'

], function (

requirejs,

$,

moment,

Jupyter,

events,

codecell

) {

'use strict';

var mod\_name = 'Lighter';

var log\_prefix = '[' + mod\_name + ']';

var CodeCell = codecell.CodeCell;

var blink=0;

var plugin\_status = true;

var stack = [];

var params = {

colors:{

successful\_exec: '#00BB00',

error\_exec: '#cc0000',

execution: '#42A5F5',

waiting: '#FFFF66'

},

toogle\_plugin\_hotkey : 'Alt-N',

button\_enable : true

};

function patch\_CodeCell\_get\_callbacks () {

console.log(log\_prefix, 'patching CodeCell.prototype.get\_callbacks');

var old\_get\_callbacks = CodeCell.prototype.get\_callbacks;

CodeCell.prototype.get\_callbacks = function () {

var callbacks = old\_get\_callbacks.apply(this, arguments);

var cell = this;

var prev\_reply\_callback = callbacks.shell.reply;

callbacks.shell.reply = function (msg) {

console.log(msg.msg\_type);

if (msg.msg\_type === 'execute\_reply') {

setTimeout( function(){

if ($.ui !== undefined) {

stack.shift();

if(stack.length !=0){

stack[0].element.find('.input\_area').stop(true,false);

create\_blink(0);

};

var input\_area = cell.element.find('.input\_area');

input\_area[0].title=0;

var color;

input\_area.stop(true, false);

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

if((cell.output\_area.outputs === undefined ||

cell.output\_area.outputs.length == 0) ||

cell.output\_area.outputs[0].output\_type != 'error') color = params.colors.successful\_exec

else color = params.colors.error\_exec;

if(plugin\_status)input\_area.show(0).effect('highlight', {color: color});

}

}, 5 );

}

return prev\_reply\_callback(msg);

};

return callbacks;

};

}

function create\_blink (num){

var fadeto = 0.1;

if(num==0) fadeto = 0.35;

var input\_area = stack[num].element.find('.input\_area');

function initpulse(){

input\_area.fadeTo(1480, 0.03,after1)

}

function after1(){

if(input\_area[0].title == 1){

var color;

if(num==0)color = params.colors.execution

else color = params.colors.waiting;

input\_area[0].style.backgroundColor = color;

pulsatingIn();

} else{

input\_area.stop(true,false);

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

}

}

function pulsatingIn(){

if(input\_area[0].title == 1)

input\_area.fadeTo(2500, 1, pulsatingOut)

else{

input\_area.stop(true,false);

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

}

}

function pulsatingOut(){

if(input\_area[0].title == 1)

input\_area.fadeTo(1400, fadeto, pulsatingIn)

else{

input\_area.stop(true,false);

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

}

}

if(input\_area[0].title == 1)

initpulse()

else{

input\_area.stop(true,false);

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

}

}

function excute\_codecell\_callback (evt, data) {

if(!plugin\_status) return;

var cell = data.cell;

cell.element.find('.input\_area')[0].title=1;

for(var i=0; i<stack.length;i++){

if(cell.cell\_id == stack[i].cell\_id){

stack.splice(i,1);

if(i==0){

stack[0].element.find('.input\_area')[0].style.backgroundColor = params.colors.execution;

}

break;

}

}

stack.push(cell);

create\_blink(stack.length-1);

}

var update\_params = function() {

var config = Jupyter.notebook.config;

for (var key in params) {

if (config.data.hasOwnProperty(key) ){

params[key] = config.data[key];

}

}

};

var toggle\_all = function() {

plugin\_status = !plugin\_status;

change\_button\_color();

if(plugin\_status){

console.log(JSON.stringify(stack));

for(var i in stack ) {

var input\_area = stack[i].element.find('.input\_area');

input\_area[0].title=1;

create\_blink(i);

}

} else{

for(var i in stack){

var input\_area = stack[i].element.find('.input\_area');

input\_area.stop(true,false);

input\_area[0].title=0;

input\_area[0].style.opacity = 1;

input\_area[0].style.backgroundColor = "#F7F7F7";

}

}

};

var change\_button\_color = function() {

var button = document.getElementsByClassName("fa fa-bomb")[0].parentNode;

if(plugin\_status){

button.style.backgroundColor = "#baffc9";

} else{

button.style.backgroundColor = "#ffffff";

}

};

// define action, register with ActionHandler instance

var prefix = 'auto';

var action\_name = 'toogle\_plugin';

var action = {

icon: 'fa fa-bomb',

help: 'Turn on/off highlighting',

help\_index : 'zz',

id: 'toggle\_plugin',

handler: toggle\_all

};

var action\_full\_name;

function load\_jupyter\_extension () {

update\_params();

action\_full\_name = Jupyter.keyboard\_manager.actions.register(action, action\_name, prefix);

// create toolbar button

if(params.button\_enable) Jupyter.toolbar.add\_buttons\_group([action\_full\_name]);

change\_button\_color();

// define hotkey

if (params.toogle\_plugin\_hotkey) {

Jupyter.keyboard\_manager.edit\_shortcuts.add\_shortcut(

params.toogle\_plugin\_hotkey, action\_full\_name);

Jupyter.keyboard\_manager.command\_shortcuts.add\_shortcut(

params.toogle\_plugin\_hotkey, action\_full\_name);

}

// try to load jquery-ui

if ($.ui === undefined) {

requirejs(['jquery-ui'], function ($) {}, function (err) {

// try to load using the older, non-standard name (without hyphen)

requirejs(['jqueryui'], function ($) {}, function (err) {

console.log(log\_prefix, 'couldn\'t find jquery-ui, so no animations');

});

});

}

Jupyter.notebook.config.loaded.then(function on\_config\_loaded () {

$.extend(true, params, Jupyter.notebook.config.data[mod\_name]);

}, function on\_config\_load\_error (reason) {

console.warn(log\_prefix, 'Using defaults after error loading config:', reason);

}).then(function () {

patch\_CodeCell\_get\_callbacks();

events.on('execute.CodeCell', excute\_codecell\_callback);

}).catch(function on\_error (reason) {

console.error(log\_prefix, 'Error:', reason);

});

}

return {

load\_jupyter\_extension : load\_jupyter\_extension,

load\_ipython\_extension : load\_jupyter\_extension

};

});

**Markdown файл – readme.md**

Markdown — облегчённый язык разметки, созданный с целью написания наиболее читаемого и удобного для правки текста, но пригодного для преобразования в языки для продвинутых публикаций (HTML, Rich Text и других).

Markdown очень удобен для составления красивого описания. Но мне было лень, и я просто добавил туда две строчки:

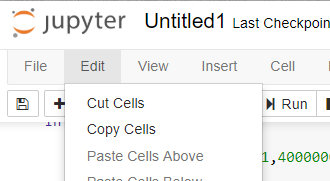
Подсветка выпоняемых ячеек

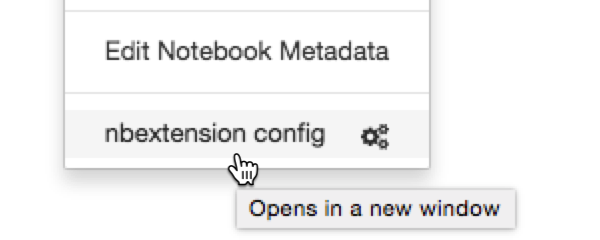
**============**

Описание расширения

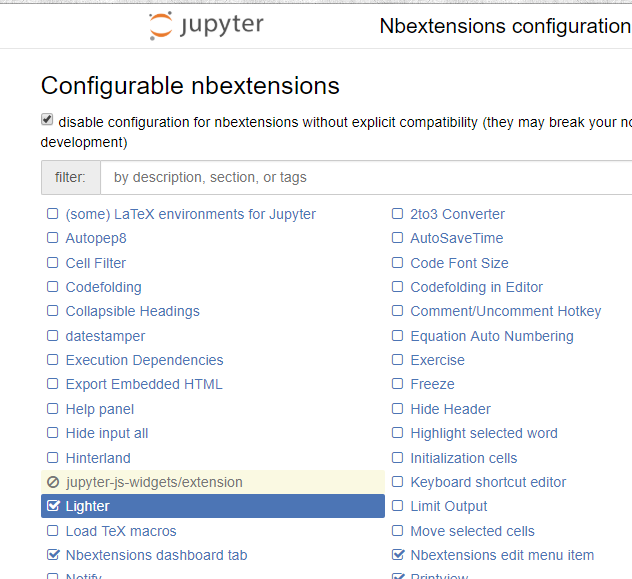
**Кратко: Lighter** – расширение для подсветки ячеек.

Заходим в меню edit на странице Jupyter Notebook и нажимаем на nbextensions config для включения нашего расширения:



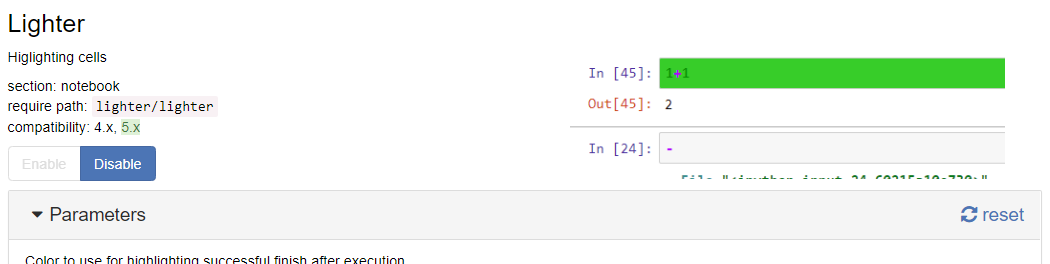


Попадаем в конфигуратор расширений. Находим наш плагин с названием Lighter и включаем его.

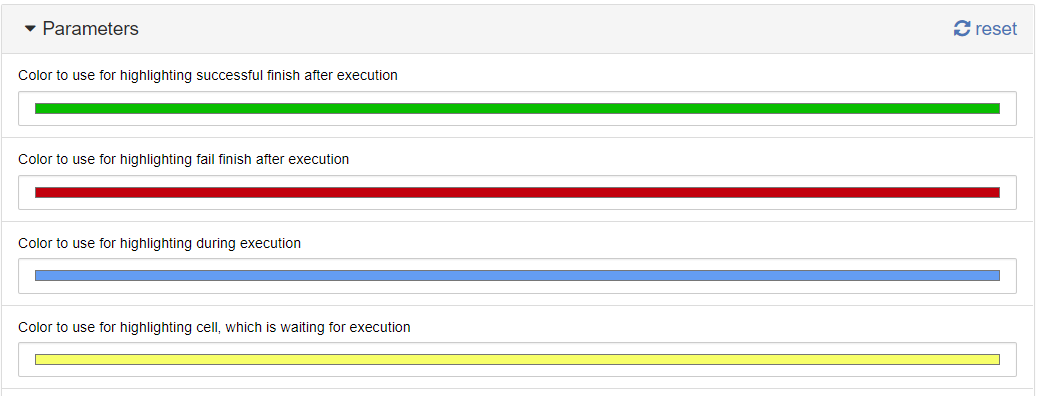


Ниже можно посмотреть описание расширения, которое настраивалось с помощью YAML файла.

Краткое описание и иконка расширения:

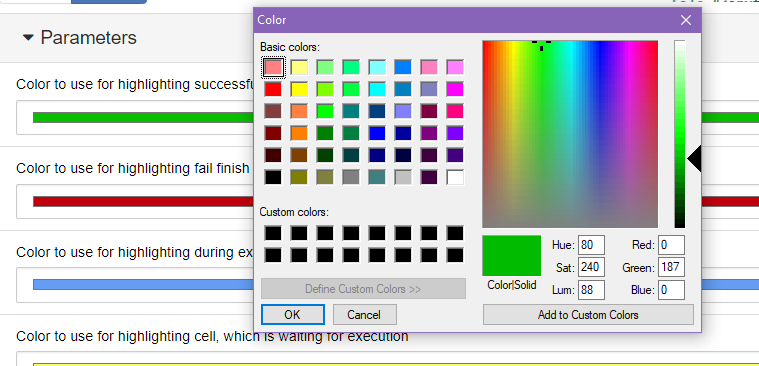


Параметры выбора цвета подсвечивания ячеек в разных случаях

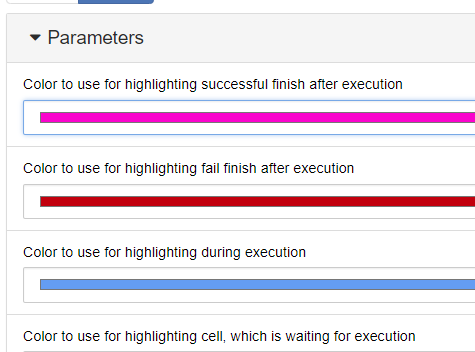


1. Цвет ячейки после успешного выполнения
2. Цвет ячейки после выполнения с ошибкой
3. Цвет ячейки при выполнении
4. Цвет ячейки, которая находится в очереди на выполнение

Также можно устанавливать свой цвет, какой захочешь, используя палитру.



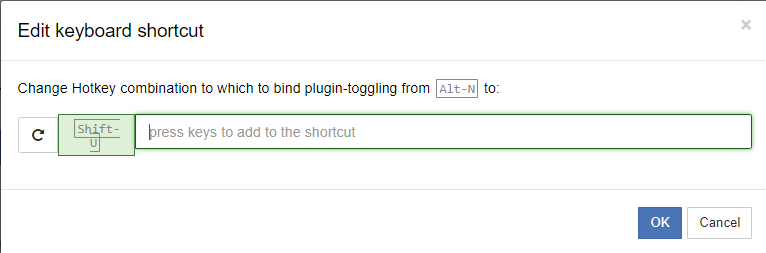
Изменим цвет ячейки после успешного выполнения на розовый:



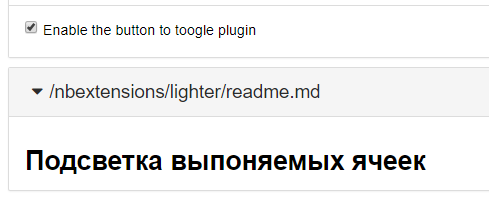
Далее идёт настройка горячей клавиши и добавление/удаление кнопки с панели:



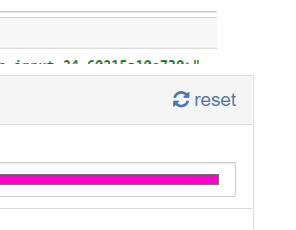
Сейчас на горячей клавише стоит комбинация Alt-N. Изменим её на нажатие кнопки Shift-U:



Вконце прикреплен readme.md файл с описанием расширения.

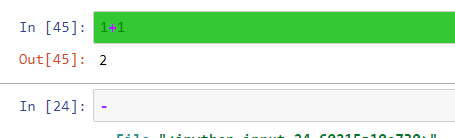


Чтобы сбросить все настройки к дефолтным, нажмём кнопку reset

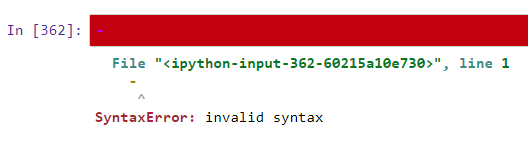


Теперь возвращаемся к Jupyter Notebook и проверим работоспособность расширения в действии. Для этого вначале нужно перезагрузить тетрадку, чтобы все изменения пришли в силу.

Успешное выполнение ячейки:



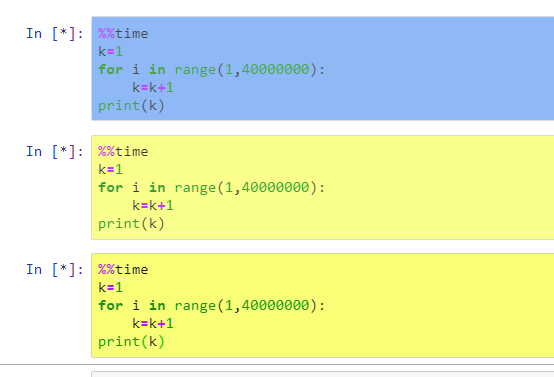
Выполнение с ошибкой:



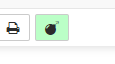
Ячейка во время выполнения:



Ячейки во время выполнения с очередью:



Кнопка включение/выключение плагина:



**История коммитов**

