

ПРИНЦИПЫ РАБОТЫ В СРЕДЕ ПРОГРАММИРОВАНИЯ РУСНАРМ

 Занятие №5, МОДУЛЬ 1

Основы PyCharm, особенности данной среды разработки



Цель занятия: Изучение основ среды PyCharm.

- 1. Среда разработки** – система, в которой реализуется процесс создания и конфигурирования приложений, которые могут использоваться в различных областях.
- 2. Приложение** - код, состоящий из одного или нескольких системных файлов, который вместе образуют единую систему.



1. Что такое кодирование?

2. Какие компоненты есть у ПК?

3. Что такое ввод/вывод данных?

4. Какими бывают программы?

5. Что такое программа?

Среда PyCharm



Начать можно с того, что **PyCharm** обладает максимальной продуктивностью, а значит может позаботиться о рутинных задачах, чтобы архитектор-разработчик смог сосредоточиться на более важных вещах.

Работая в PyCharm, разработчики экономят время, так как для большинства задач не нужно отрывать руки от клавиатуры. В тоже время, PyCharm все знает о коде. Среда PyCharm имеет встроенный редактор, который может помочь пользователю в нахождение синтаксических, логических и прочих ошибок. Данная среда удобно подсвечивает ключевые слова, тела функций / циклов и прочее.

Умный редактор



Умный механизм анализа и редактирования кода обеспечивает точное автодополнение, поиск ошибок и быстрые исправления, удобную навигацию по коду и другие полезные функции:

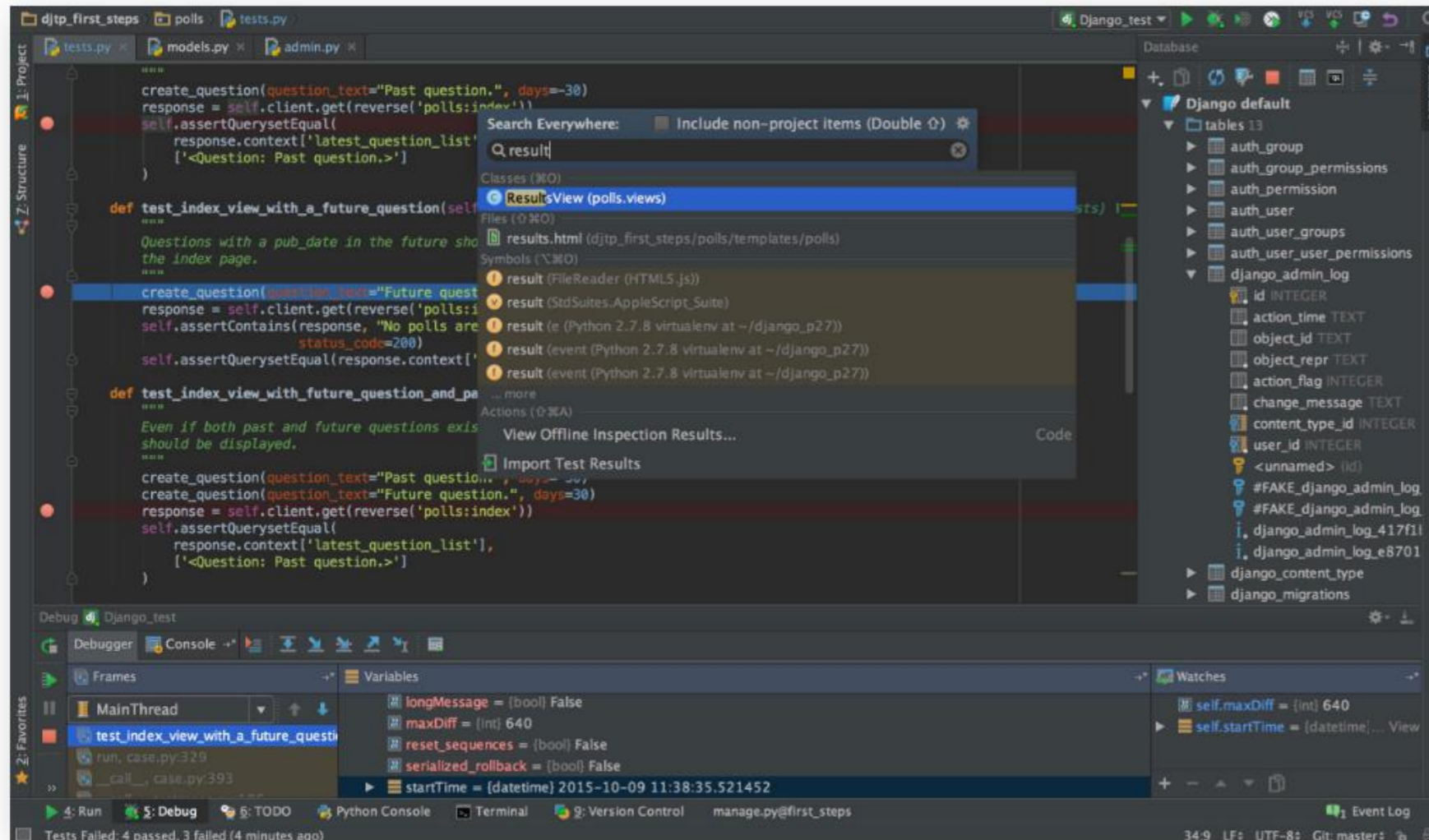
The screenshot shows a code editor with a Python file named `tests.py`. The code includes several test methods for a Django application. An autocomplete popup menu is visible over the `self.test` line, listing various methods and attributes of the `QuestionViewTests` class, such as `test_index_view_with_a_future_question`, `test_index_view_with_a_past_question`, and `test_index_view_with_future_question_and_past_question`. The editor also shows a status bar at the bottom with the message: "Statement seems to have no effect. Unresolved attribute reference 'test' for class 'QuestionViewTests'."

```
20 """
21 response = self.client.get(reverse('polls:index'))
22 self.assertEqual(response.status_code, 200)
23 self.assertContains(response, "No polls are available.")
24 self.assertQuerysetEqual(response.context['latest_question_list'], [])
25 self.test
26
27 def test_index_view_with_a_future_question(self):
28     """
29     Questions with a pub_date in the future should not be displayed on
30     the index page.
31     """
32     create_question(question_text="Future question.", days=30)
33     response = self.client.get(reverse('polls:index'))
34     self.assertContains(response, "No polls are available.",
35                       status_code=200)
36     self.assertQuerysetEqual(response.context['latest_question_list'], [])
37
38 def test_index_view_with_future_question_and_past_question(self):
39     """
40     Even if both past and future questions exist, only past questions
41     should be displayed.
42     """
43     create_question(question_text="Past question.", days=-30)
44     create_question(question_text="Future question.", days=30)
45     response = self.client.get(reverse('polls:index'))
46     self.assertQuerysetEqual(
47         response.context['latest_question_list'],
48         ['<Question: Past question.>']
49     )
50
51 def test_index_view_with_two_past_questions(self):
52     """
53     """
54     create_question(question_text="Past question.", days=-30)
55     create_question(question_text="Past question.", days=-30)
56     response = self.client.get(reverse('polls:index'))
57     self.assertQuerysetEqual(
58         response.context['latest_question_list'],
59         ['<Question: Past question.>']
60     )
61
62 def test_index_view_with_two_past_questions(self):
63     """
64     """
```

УМНЫЙ редактор



Все это дополняется повышением качества кода. PyCharm помогает писать красивый код, который легко поддерживать. IDE контролирует качество кода с помощью проверок соответствия требованиям PEP-8, умных рефакторингов и множества инспекций, а также оказывает помощь при тестировании:



Умный редактор



С помощью умного детектора дубликатов PyCharm проверяет код на наличие дублирующихся фрагментов. IDE предложит список фрагментов, которые следует преобразовать, а рефакторинги помогут избавиться от повторяющегося кода.

Разработчик также может выбрать один из доступных стилей кода, разработанного для каждого из поддерживаемых языков. Помимо этого, в данной профессиональной среде разработки предусмотрены: сворачивание блоков кода, автоматическая расстановка скобок и кавычек, а также подсветка парных скобок и т.д.

Удобная **навигация**



Функция *Search Everywhere* поможет архитекторам-разработчикам найти класс, файл, действие или элемент интерфейса IDE. Вызвать данную функцию можно двойным нажатием клавиши **Shift**, а затем вызвать поисковый запрос:

```
main.py x
507/
508 # dy ranges from -1 to 1 and is -1 when looking straight down and 1 when
509 # looking straight up.
510 dy = math.sin(math.radians(y))
511 dx = math.cos(math.radians(x - 90)) * m
512 dz = math.sin(math.radians(x - 90)) * m
513 return (dx, dy, dz)
514
515 def get_motion_vector(self):
516     """ Returns the current motion vector indicating the
517     player.
518
519     Returns
520     _____
521     vector : tuple of len 3
522         Tuple containing the velocity in x, y, and z respec
523
524     """
525     if any(self.strafe):
526         x, y = self.rotation
527         strafe = math.degrees(math.atan2(*self.strafe))
528         y_angle = math.radians(y)
529         x_angle = math.radians(x + strafe)
530         if self.flying:
531             m = math.cos(y_angle)
```

Search Everywhere: Include non-project items (Double ⇧) ⚙

Q cub

Classes (⌘O)

- 🔍 cubic_centimeters (StdSuites.AppleScript_Suite)
- 🔍 cubic_feet (StdSuites.AppleScript_Suite)
- 🔍 cubic_inches (StdSuites.AppleScript_Suite)
- 🔍 cubic_meters (StdSuites.AppleScript_Suite)
- 🔍 cubic_yards (StdSuites.AppleScript_Suite)
- 🔍 c_ubyte (ctypes)

Files (⇧⌘O)

- 🔍 test_cursor_pget_bug.py (/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/bsddb/test)

Symbols (⇧⌘O)

- 🔍 cube_vertices (main)

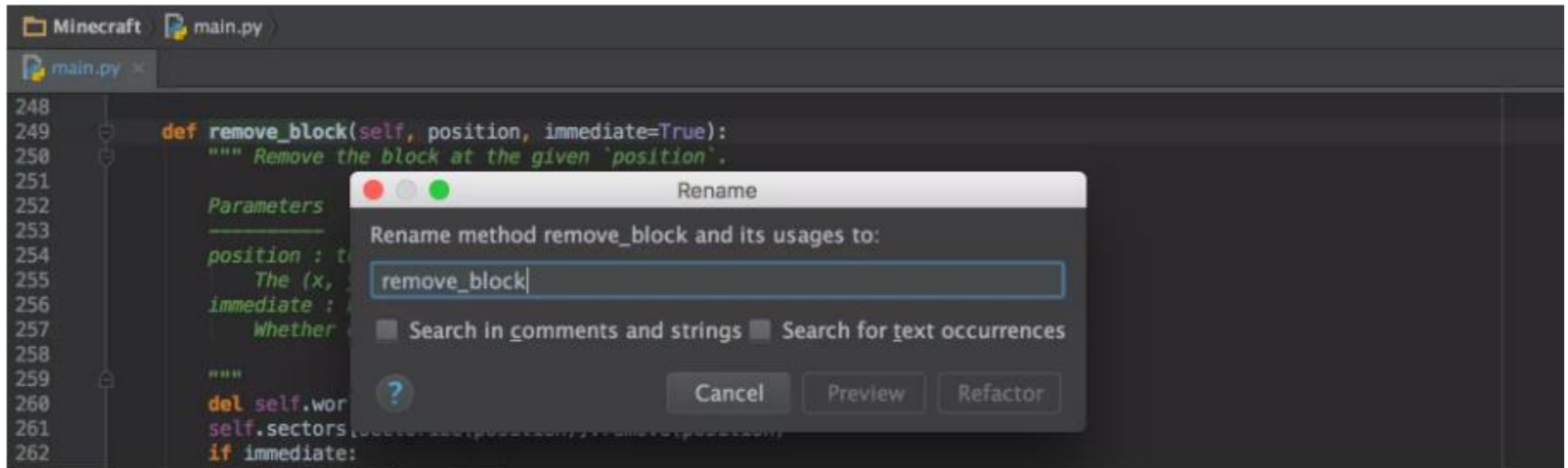
Actions (⇧⌘A)

- Cut Line Backward

Быстрые и безопасные рефакторинги



Рефакторинги **Rename** и **Move** применимы для файлов, функций, констант, классов, свойств, методов, параметров локальных и глобальных переменных:



Быстрые и безопасные рефакторинги



Для эффективной реорганизации кода доступны и другие рефакторинги: изменения сигнатуры, конвертация и пакет/модуль Python, создания функции верхнего уровня и т.д.

В разделе «**Документация**» разработчики могут посмотреть определения объектов и читать документацию, не покидая IDE:

```
126 # A Batch is a collection of vertex lists for batched rendering.
127 self.batch = pygame.graphics.Batch()
128
129 # A TextureGroup manages an OpenGL texture.
130 self.group = TextureGroup(image.load(TEXTURE_PATH).get_texture())
131
132 # A mapping from position to the t
133 # This defines all the blocks that
134 self.world = {}
135
136 # Same mapping as `world` but only
137 self.shown = {}
138
139 # Mapping from position to a pygame
140 self._shown = {}
141
142 # Mapping from sector to a list of positions inside that sector.
143 self.sectors = {}
144
```

Documentation for load(filename, file=None, decoder=None)

< Python 2.7.10 virtualenv at ~/minecraftvenv >

```
def load(filename, file=None, decoder=None)
Inferred type: (filename: Any, file: Any, decoder: Any) -> AbstractImage
```

Load an image from a file.

note: You can make no assumptions about the return type; usually it will be ImageData or CompressedImageData, but decoders are free to return any subclass of AbstractImage

Основные преимущества



Давайте еще раз выделим **основные преимущества** данной профессиональной среды:

- **Умный редактор.** В PyCharm легко редактировать код благодаря автодополнению и проверке кода, подсветке ошибок и быстрым исправлениям, а также автоматическому рефакторингу и удобной навигации.
- **Поддержка веб-фреймворков.** PyCharm поддерживает современные фреймворки для веб-разработки: Django, Flask, Google App Engine, Pyramid и web2py.

Основные преимущества



- **Поддержка научных вычислений.** В PyCharm разработчики могут работать с Jupyter-ноутбуками, запускать команды в интерактивной консоли Python, подключать библиотеки Anaconda, а также работать с другими библиотеками для научных вычислений и анализа данных, включая Matplotlib и Numpy.
- **Разработка на разных языках.** Помимо Python, PyCharm поддерживает JavaScript, CoffeeScript, TypeScript, Cython, SQL, HTML/CSS, языки шаблонов, AngularJS, Node.js и другие технологии.

Основные преимущества



- **Возможности удаленной разработки.** В PyCharm разработчики могут запускать, отлаживать, тестировать и развертывать приложения на удаленных хостах или виртуальных машинах с помощью удаленных интерпретаторов, встроенного SSH-терминала и интеграции с Docker и Vagrant.
- **Встроенные инструменты для разработчиков.** PyCharm предоставляет встроенный отладчик и инструмент запуска тестов, профилировщик Python, встроенный терминал, инструмент для работы с базами данных и интеграцию с популярными системами контроля версий.

Форматирование кода.

Мультикурсоры



DataGrip автоматически отформатирует *SQL*-код в соответствии с настройками стиля. Разработчикам не нужно тратить много времени на форматирование старого кода вручную - он может воспользоваться комбинацией клавиш **Ctrl + Alt + L**:

```
select *
from (select *
      from actor
           join film_actor fa on actor.actor_id = fa.actor_id
           join film f on fa.film_id = f.film_id
           join film_category f2 on f.film_id = f2.film_id
      where fa.actor_id > 12) x
order by x.film_id
```

Formatted 7 lines
Show reformat dialog: `\u2715\u2191\u2715L`

Форматирование кода.

Мультикурсоры



Мульти-курсоры - эффективный способ писать или менять запросы. Для того, чтобы расставить мульти-курсоры в *DataGrip*, нужно дважды нажать на клавишу **Ctrl** и, удерживая данную клавишу нажатой, использовать клавиши стрелок вверх/вниз. Также доступен текстовый поиск. После нахождения нужного фрагмента текста, нужно нажать на иконку *Select All Occurrences*, которая находится справа от поля ввода на панели поиска:

```
Postgres in Docker x
actor_id
↑ ↓ ALL + II - II II
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('CHRIST')
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('ZERO',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('KARL',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('UMA',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('VIVIEN',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('CUBA',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('FRED',
INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES ('HELEN',
```


Шаблоны кода.

Локальная история



Архитекторы-разработчики могут использовать встроенные шаблоны кода, чтобы не писать все время одно и то же. Для отображения выпадающего списка шаблонов, нужно нажать комбинацию клавиш **Ctrl + J**:

```
Postgres in Docker x
```

Tx: Auto

```
postgres.<no schema>
```

```
SELECT *  
FROM actor a  
WHERE a.actor_id = 10;  
  
INSERT INTO | () VALUES ();
```

actor (postgres.public)	Postgres in Docker
film (postgres.public)	Postgres in Docker
film_category (postgres.public)	Postgres in Docker
actor_1 (postgres.public)	Postgres in Docker
actor_actor_id_seq (postgres.public)	Postgres in Docker
address (postgres.public)	Postgres in Docker

Шаблоны кода.

Локальная история



В той ситуации, если пользователю нужно сравнить версии, то ему в этом может помочь интерфейс *Diff Viewer*:

```
Yesterday 16:12 1 file console.sql
Yesterday 16:12 1 file console.sql
Yesterday 16:09 1 file console.sql
Yesterday 15:48 1 file console.sql
Yesterday 15:35 console.sql
Yesterday 15:34 console.sql
Yesterday 15:33 console.sql
Yesterday 15:33 console.sql
Yesterday 15:32 console.sql
Yesterday 15:32 console.sql
Yesterday 12:34 1 file console.sql
```

Side-by-side viewer Do not ignore >> 1 difference

Yesterday 15:35 - console.sql (Re...)	Current
1 select *	1 select *
2 from (select *	2 from actor;
3 from actor)	3
	4 select *
	5 from actor a
	6 join film_actor fa on a.a
	7 join film f on fa.film_id
	8 join film_category f2 on
	9
	10 select *
	11 from film_category as actor
	12
	13 insert into actor (first_na
	14 values ('JAMES', 'KERRY', c
	15
	16

Revert
Create Patch...
Help

История буфера обмена.

Проверка орфографии



В *DataGrip* хранится история буфера обмена. IDE записывает все, что пользователь копирует в буфер и хранит это, пока программа не будет закрыта окончательно:

```
1 https://www.dropbox.com/s/r16qnm4jbtuud5s/Screenshot%202018-01-23%2012.23.04.png?
2 https://www.dropbox.com/s/nvtb37hlju7ye3i/Screenshot%202018-01-23%2012.22.56.png?
3 [{"actor_id": 10, "first_name": "CHRISTIAN", "last_name": "GABLE"}
4 https://www.dropbox.com/s/ri32n4nyiw0x3t/Screenshot%202018-01-23%2012.22.21.png?
5 https://www.dropbox.com/s/e9k9y4fdpcq8hvu/Screenshot%202018-01-23%2012.16.58.png?
6 actor_id
7 INSERT INTO public.actor (actor_id, first_name, last_name, last_update) VALUES (1

[
  {
    "actor_id": 10,
    "first_name": "CHRISTIAN",
    "last_name": "GABLE",
    "last_update": "2006-02-15 04:34:33.000000"
  },
  {
    "actor_id": 11,
    "first_name": "ZERO",
```

История буфера обмена.

Проверка орфографии



DataGrip проверяет орфографию везде: в текстовых файлах и в данных таблиц.
Для того, чтобы исправить опечатку, пользователю нужно нажать **Alt + Enter**:

```
Docker SQL Server x
```

testdb ▼

```
select  
  actor.actor_id,  
  actor.first_name as 'FirstNaem',  
  actor.last_name,  
  actor.last_update  
from actor
```

- Nae
- Nam
- Name
- Na 's
- Nab

Языковые вставки. Динамический SQL



По умолчанию, *DataGrip* может определить только два формата литералов: *XML* и *JSON*.
Но можно и подсказать IDE, что внутри строки - код:

```
CREATE TABLE Simpsons
(
  json_data JSON
);

INSERT INTO Simpsons (json_data)
VALUES ('{
  "first_name": "Homer",
  "last_name": "Simpson",
  "age": "38",
  "address": {
    "street_address": "742 Evergreen Terrace",
    "city": "Springfield",
    "state": ""
  }
}');
```

Языковые вставки. Динамический SQL



Для динамического SQL внутри строки, в это же время, заработают автодополнение, инспекции и другие привычные функции.

Посмотрим на реализацию динамического SQL в данной среде:

```
DECLARE @myQ AS NVARCHAR(MAX)

--language=TSQL
SET @myQ =
'SELECT *
FROM Authors a
WHERE a.'

EXEC sp_
```

FirstName (a)	nvarchar
ID (a)	int
LastName (a)	nvarchar
\$action	string
\$IDENTITY	string
\$ROWGUID	string

Press ^. to choose the selected (or first) suggestion and insert a do

Языковые вставки. Динамический SQL



Как и в *Jupyter*, закомментировать код в *DataGrip* очень просто. Для этого нужно использовать клавиши для *построчного комментирования* или *блоками*, если это поддерживается SQL-диалектом:

```
SELECT *  
FROM actor  
  JOIN film_actor fa ON actor.actor_id = fa.actor_id  
  JOIN actor a ON fa.actor_id = a.actor_id  
—  JOIN actor a2 ON fa.actor_id = a2.actor_id
```

```
— SELECT *  
— FROM actor  
—  JOIN film_actor fa ON actor.actor_id = fa.actor_id  
—  JOIN actor a ON fa.actor_id = a.actor_id  
—  JOIN actor a2 ON fa.actor_id = a2.actor_id
```

Языковые вставки. Динамический SQL



Кроме этого, в текстовом редакторе платформы **IntelliJ** встроена возможность навигации по коду. Посмотрим на основные комбинации клавиш: «**Ctrl + Alt + <- / ->**» или «**Ctrl + Shift + Backspace**». Посмотрим на пример работы с текстовым редактором платформы **IntelliJ**:

```
Docker SQL Server x UNIT-781 x  
Tx: Auto testdb  
select *  
from actor;  
insert into city (city, country_id)  
values ('New Brighton', 34);
```


Языковые вставки. Динамический SQL



Данные операции работают и с частями кода:

```
Docker SQL Server x
Tx: Auto
select *
from actor;

insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
```

```
Docker SQL Server x
Tx: Auto

insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
select *
from actor;
insert into city (city, country_id)
values ('New Brighton', 34);
insert into city (city, country_id)
values ('New Brighton', 34);
```

Языковые вставки. Динамический SQL



Кроме этого, в текстовом редакторе платформы **IntelliJ** встроена возможность навигации по коду. Посмотрим на основные комбинации клавиш: «**Ctrl + Alt + <- / ->**» или «**Ctrl + Shift + Backspace**». Посмотрим на пример работы с текстовым редактором платформы **IntelliJ**:

```
Docker SQL Server x UNIT-781 x  
Tx: Auto testdb  
select *  
from actor;  
insert into city (city, country_id)  
values ('New Brighton', 34);
```

ОСНОВНЫЕ ОТЛИЧИЯ



Также рассмотрим **основные отличия** *Community* и *Pro* версий:

- *Community* не поддерживает *JavaScript*, *CSS* и другие веб-технологии. В данной версии можно работать только с языком *Python* и его вариациями, а также с *HTML*, *XML*, *JSON* и некоторыми другими форматами.
- То же самое можно сказать и о поддержке популярных фреймворков. **PyCharm Community Edition** поддерживает только фреймворки для *Python*, а *Pro*-версия - еще и *React*, *Angular* и другие инструменты веб-разработки, ориентированные на *JavaScript*.

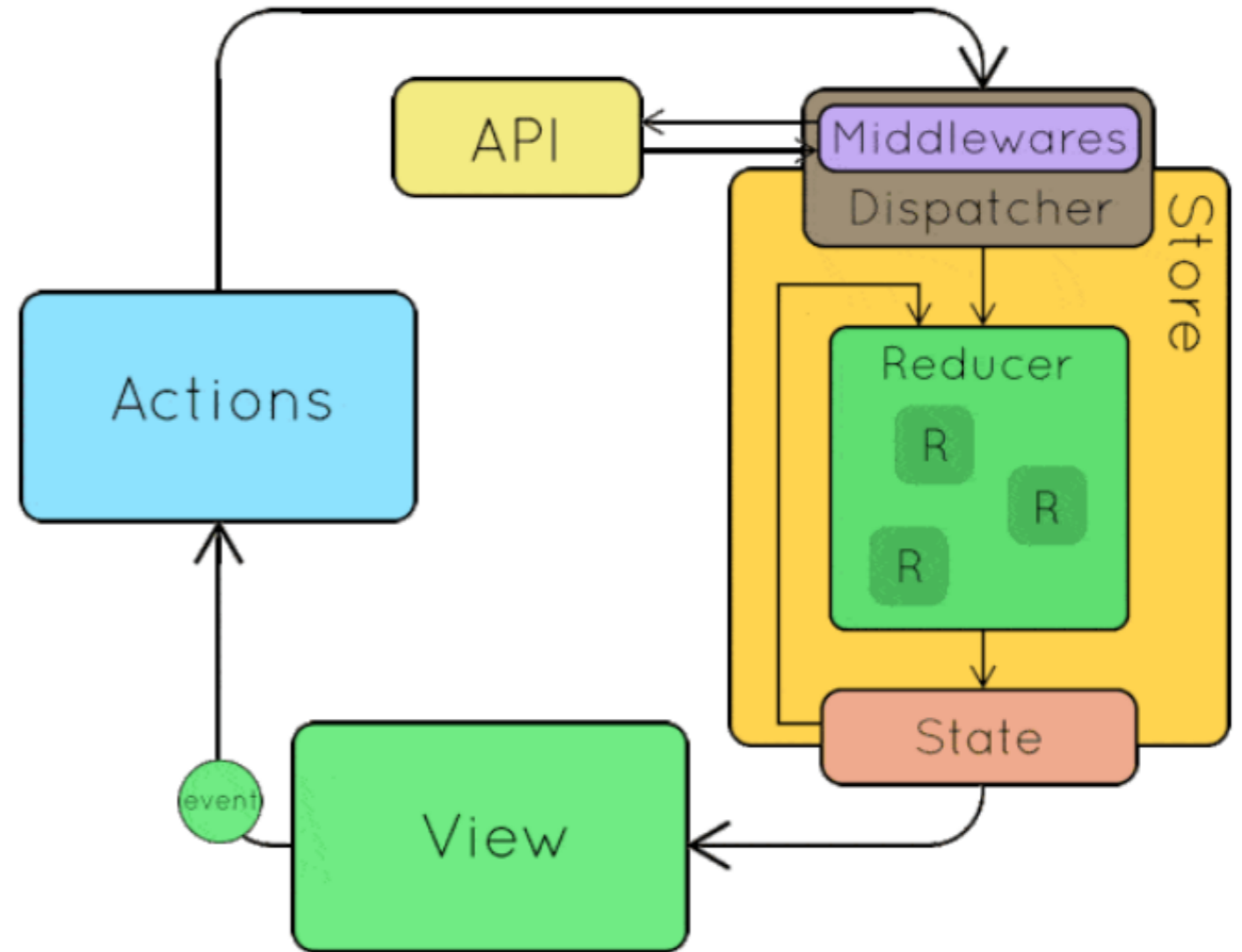
ОСНОВНЫЕ ОТЛИЧИЯ



- В *Community*-версии нет встроенных инструментов для работы с базами данных, в то время как в *Pro*-версии они есть.
- Инструменты для развертывания и контроля версий в *Community* устанавливаются отдельно как плагины, а в *Pro* они предустановлены.
- Возможности совместной разработки в бесплатной версии ограничены: не более трех человек и сессии не больше 30 минут.

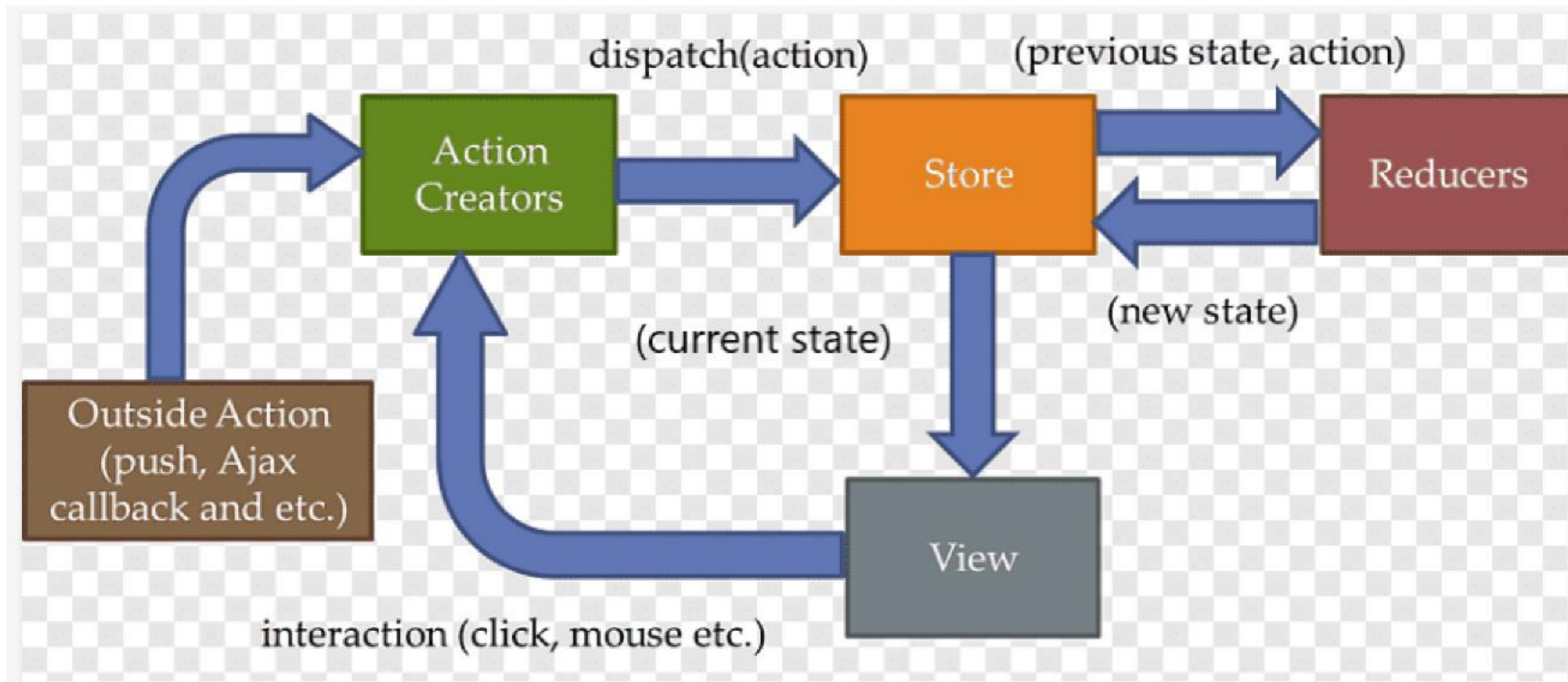


Покажем как управлять состоянием в *Angular* по мере роста простого приложения для произвольной области:





Ниже приведена блок-схема данных **GitHub** на **React**





Full-stack разработка с *Python* - широкая область, где среда **PyCharm** может помочь:

- Многофункциональный *HTTP*-клиент для автоматизированного тестирования
- Поддержка фреймворка **web2py**: особый тип конфигурации *run/debug*, поддержка языка шаблонов, навигация между представлениями и шаблонами и шаблон проекта **web2py**
- Поддержка **Google App Engine (GAE)**: настройка параметров GAE прямо в IDE, загрузка приложений из меню инструментов, просмотр файлов в логе, специальная консоль для запуска команд *appcfg.py* и шаблон проекта GAE
- Языковые вставки для встраивания поддержки SQL в строки на *Python* с автодополнением не только для команд SQL, но и для схемы заданного источника данных.

Введение в Spyder



Посмотрим на пример работы в *Spyder*:

The screenshot displays the Spyder Python IDE interface. The left pane shows a Python script named 'Request.pyw' with code for generating HTML from a database. The right pane shows a variable manager table with columns 'Имя', 'Тип', 'Размер', and 'Значени'. Below the table is a console window showing the execution of the script and a NameError.

```
1161     pass
1162
1163     printer = QtPrintSupport.QPrinter()
1164
1165     html = '<h1 style="text-align: center;">Заявка №' + str(
1166           rowP[1]) + '</h1> <br>'
1167     # html = html + '<table style="border-color: #000000;" border="2"
1168     # cellspacing="1" cellpadding="1">'
1169     html = html + '<table style="border-color: #000000;" border="1"
1170           'cellspacing="0" cellpadding="2">'
1171     html = html + '<tbody>'
1172
1173     html = html + '<tr>' + '<td>ID</td> <td>' + str(rowP[0]) + '</td>'
1174     html = html + '<tr>' + '<td>Номер</td> <td>' + str(rowP[1]) + '</td>'
1175
1176     # Получаю тип из связанной таблицы
1177     s1 = 'SELECT * FROM types WHERE (id LIKE "' + str(rowP[2]) + '")'
1178     CurPrint.execute(s1)
1179     recordsP1 = CurPrint.fetchall()
1180     for rowP1 in recordsP1:
1181         pass
1182
1183     html = html + '<tr>' + '<td>Тип</td> <td>' + str(rowP1[1]) + '</td>'
1184     vazh = rowP[0]
1185     if vazh == 0:
1186         vazhnost = 'Обычная'
1187     elif vazh == 1:
1188         vazhnost = 'Важная'
1189     else:
1190         vazhnost = 'Неизвестно'
1191     sta = rowP[0]
1192     if sta == 0:
1193         status = 'В работе'
1194     elif sta == 1:
1195         status = 'Отменена'
1196     elif sta == 2:
1197         status = 'Выполнена'
1198     else:
1199         status = 'Неизвестно'
1200     html = html + '<tr>' + '<td>Значимость</td> <td>' + vazhnost + '</td>'
1201     html = html + '<tr>' + '<td>Статус</td> <td>' + status + '</td> </tr>'
```

Имя	Тип	Размер	Значени
app	SpyderQApplication	1	SpyderQAp
App	SpyderQApplication	1	SpyderQAp
cifra	int	1	218
cifra2	int	1	97
login	Login	1	Login obj
py3	bool	1	True
qApp	QtWidgets.QApplication	1	QApplicat

```
<module>
MainTableContent()
File "D:\Request\Request.pyw", line 495, in
MainTableContent
while (window.requestTable.rowCount() > 0):
NameError: name 'window' is not defined

In [2]: runfile('D:/Request/Request.pyw',
wdir='D:/Request')

In [3]: runfile('D:/XmlSitemapGenerator/Next/
sitemap.py', wdir='D:/XmlSitemapGenerator/Next')

In [4]:
```


ПРАКТИЧЕСКИЕ ЗАДАЧИ



Задача 1



Создать программу в среде PyCharm, в которой вводятся с клавиатуры три переменные с типом данных **int**: **'a'**, **'b'** и **'c'**. Затем в переменную **'z'** записывается сумма значений данных переменных. Вывести на консоль значения введенных переменных и сумму.

Решение



Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:

```
pythonProject3 - main.py
pythonProject3 > main.py
1 # Ввод значений
2 a = int(input('Введите a: '))
3 b = int(input('Введите b: '))
4 c = int(input('Введите c: '))
5 # вычисление суммы
6 z = a + b + c
7 # вывод данных
8 print('Значение a:', a)
9 print('Значение b:', b)
10 print('Значение c:', c)
11 print('Сумма:', z)
```

Run: main x

```
↑ Введите a: 5
↓ Введите b: 10
↕ Введите c: 15
Значение a: 5
Значение b: 10
Значение c: 15
Сумма: 30
```

Version Control Python Packages TODO Python Console Problems Terminal Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes //

Задача 2



Создать программу в среде PyCharm, в которой вводятся с клавиатуры три переменные с типом данных **float**: 'x', 'y' и 'z'. Затем в переменную 's' записывается произведений значений данных переменных. Вывести на консоль значения введенных переменных и произведение.

Решение



Решение.

Для решения данной задачи понадобится следующий код:

```
pythonProject3 - main.py
pythonProject3 main.py
1 # ВВОД значений
2 x = float(input('Введите x: '))
3 y = float(input('Введите y: '))
4 z = float(input('Введите z: '))
5 # вычисление произведения
6 s = x + y + z
7 # ВЫВОД данных
8 print('Значение x:', x)
9 print('Значение y:', y)
10 print('Значение z:', z)
11 print('Произведение:', s)
```

Run: main x

```
Введите x: 1.13
Введите y: 5.32
Введите z: 7.43
Значение x: 1.13
Значение y: 5.32
Значение z: 7.43
Произведение: 13.879999999999999
```

Version Control Python Packages TODO Python Console Problems Terminal Services

Тур: In word 'Произведение'



1. Для чего используется *DataGrip*, при работе с приложениями?

2. Что такое **мульти-курсоры**?

3. Какую основную функцию выполняет операция *Select All Occurences*?

4. Что такое **редактор** и какими они бывают?

5. Какой этап разработки является самым посредственным в среде PyCharm?