

**Drug prescription system**

**Meysam Khazaee Sabor**

**Dr. Saeed Parsaa**

Summer 2023

**A ‘Drug administration system’ story(User story):**

As the end user, the doctor can view patients' files, add a new patient or delete the disease by logging in to the medication system in the first section, and can also search or edit patient information. In the second part, the doctor can add, remove, edit and search for drugs. In the third section, the doctor can visit the patient. The patient is visited in two ways:

1- Visiting and prescribing based on previous versions

2- Visiting and prescribing from the list of available drugs

At each stage after selecting the drug, the doctor determines the dose of the prescribed drug and approves the drug to be added to the final prescription.

Finally, after the drugs are added, the final prescription is prepared for the patient. Now, after the doctor's approval, it is printed and stored in the database.

**User stories: Tasks**

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose. Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose. Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

برنامه نوشته شده تجویز دارو شامل چند بخش مختلف می باشد:

دکتر به عنوان کاربر نهایی با ورود به سیستم تجویز دارو در بخش اول می تواند پرونده بیماران را مشاهده کند، بیمار جدیدی را اضافه یا بیماری را حذف کند، همچنین می تواند اطلاعات بیماران را جستجو یا ویرایش کند. در بخش دوم دکتر می تواند عملیات اضافه کردن، حذف کردن، ویرایش کردن و جستجو کردن بر روی داروها را انجام دهد. در بخش سوم نیز دکتر می تواند بیمار را ویزیت کند. ویزیت بیمار به دو روش صورت می گیرد:

1. ویزیت و تجویز بر اساس نسخه های قبل
2. ویزیت و تجویز از لیست دارو های موجود

دکتر در هر مرحله بعد از انتخاب دارو دوز داروی تجویزی را تعیین می کند و دارو را برای اضافه شدن به نسخه نهایی تایید می کند.

در آخر نیز پس از اضافه شدن دارو ها، نسخه نهایی برای بیمار مورد نظر آماده می شود. حال پس از تایید دکتر، چاپ و در پایگاه داده ذخیره می شود.

**پیش نیاز های اجرایی**

برای اجرای TDD و BDD در محیط پایتون نیاز داریم کتابخانه های مورد نیاز برای روند تست را فراخوانی کنید (import). کتابخانه های مورد نیاز برای روند تست کد به زبان پایتون به شرح زیر است:

1. Pytest
2. Unittest
3. Behave
4. Monkeypatch

در صورتی که این کتابخانه ها را نصب ندارید در محیط ترمینال دستور زیر را وارد کنید تا کتابخانه ها نصب شوند:

cmd: ~ py -m pip install [package name]

e.g.

cmd: ~ py -m pip install pytest

پس از نصب کتابخانه ها یکبار محیط برنامه نویسی را راه اندازی دوباره کرده تا اگر مشکلی در نصب یا دانلود کتابخانه ها باشد معلوم شود.

* نکته حایز اهمیت این است که کتابخانه های درون فایل تست جدای از کتابخانه های درون بدنه اصلی کد هستند لذا برای اجرای درست و بدون خطای کد باید ابتدا کتابخانه های مورد نیاز را نصب کنید.
* توجه داشته باشید که برای نصب کتابخانه های جدید بایستی به اینترنت متصل باشید.
* در این گزارش برای راحتی اجرا ضمیمه های زیر نیز نصب شده اند:

1. Pylance
2. Python
3. Behave
4. Unittest
5. **توضیحات کد برنامه**

پس از نصب پیش نیاز های لازم برای برنامه، کد آماده ی اجرا می شود. در بدنه کد 8 کلاس برای انجام پروسه ی تجویز دارو به شرح زیر پیاده سازی شده اند:

* 1. کلاس load\_database: وظیفه این کلاس انجام پردازش های پایگاه داده است. بطور خلاصه این کلاس ایجاد شده است تا یک محیط همانند یک پایگاه داده را برای برنامه ایجاد کند. پایگاه داده ی ما دارای 3 موجودیت با نام های(patient\_accounts, doctors\_accounts, medicine\_list) است. هر کدام از موجودیت ها لیستی از هر شی را با ویژگی های از قبل تعریف شده را از فایلی با فرمت .txt خوانده و در خود ذخیره می کند.
  2. کلاس patient: این کلاس مربوط به موارد استفاده ی بیمار است. در این کلاس چند تابع برای انجام عملیات مربوطه تعریف شده اند.
     1. تابع exist\_patient: عملیات مربوط به وجود یا عدم وجود بیمار را برعهده دارد.
     2. تابع create\_new\_patient\_account: این تابع عملیات ایجاد بیمار جدید را انجام می دهد.
     3. تابع index\_of\_patient: این تابع شماره ایندکس بیمار را برای استفاده ذخیره می کند.
     4. تابع search\_patient\_account: این تابع عملیات جستجو میان بیماران را بر عهده دارد.
     5. تابع show\_patient\_account: این تابع پس از جستجوی بیمار مشخصات بیمار را نمایش می دهد.
     6. تابع edit\_patient\_account: این تابع عملیات ویرایش مشخصات بیمار را انجام می دهد.
     7. تابع delete\_patient\_account: این تابع عملیات حذف بیمار از پایگاه داده را انجام می دهد.
  3. کلاس medicine: این کلاس مربوط به موارد استفاده از دارو است. در این کلاس چند تابع برای انجام عملیات مربوطه تعریف شده اند.
     1. تابع exist\_medicine: عملیات مربوط به وجود یا عدم وجود دارو را برعهده دارد.
     2. تابع index\_of\_medicine: این تابع شماره ایندکس دارو را برای استفاده ذخیره می کند.
     3. تابع add\_medicine: عملیات اضافه کردن دارو را انجام می دهد.
     4. تابع edit\_medicine: این تابع عملیات ویرایش مشخصات دارو را انجام می دهد.
     5. تابع search\_medicine: این تابع عملیات جستجو میان دارو ها را بر عهده دارد.
     6. تابع show\_medicine\_info: این تابع پس از جستجوی دارو مشخصات دارو را نمایش می دهد.
     7. تابع delete\_medicine: این تابع عملیات حذف دارو را انجام می دهد.
  4. کلاس visit: این مربوط به موارد استفاده برای ویزیت بیمار است. در این کلاس چند تابع برای انجام عملیات پیاده سازی شده اند.
     1. تابع report: این تابع گزارشی از ویزیت را نشان می دهد.
     2. تابع previous\_prescriptions: این تابع اگر بیماری از قبل نسخه ای برایش تجویز شده باشد را بررسی و نمایش می دهد.
     3. تابع prescribe\_from\_medicines\_list: این تابع زمان اجرا ابتدا از کاربر ورودی می گیرد این ورودی در واقع دریافت حرف اول داروی مورد دکتر است که با وارد شدن آن تابع لیستی از دارو هایی که با حرف اول وارد شده یکسان باشد را نشان می دهد. سپس دکتر باید از بین دارو انتخاب کرده و دارو را تجویز کند سپس مقدار دوز را وارد کرد و در نهایت تجویز را تایید می کند.
  5. کلاس doctor: این کلاس مربوط به موجودیت دکتر بوده و مشخصات دکتر اعم از نام و نام خانوادگی و نام کاربری و رمز عبور را در در بر دارد.
  6. کلاس login: این کلاس وظیفه احراز هویت کاربر را دارد و هنگام اجرا در ابتدای برنامه با توجه به تابع درونی آن از کاربر نام کاربری و رمز عبور گرفته و در صورت درست وارد شدن مشخصات مراحل بعدی برنامه نمایش داده می شود.
  7. کلاس save\_database: این کلاس برای ذخیره تمام مشخصات وارد شده از کاربر برای بیمار یا دارو را بر عهده دارد. البته این کلاس وظیفه پایگاه داده را به صورت شبیه سازی انجام می دهد یعنی فایل ها در پایگاه داده واقعی ذخیره نمی شوند بلکه در مسیری تعریف شده درون تابع به فرمت .txt مشخصات را ذخیره می کند.
  8. کلاس menu: این کلاس مراحل اجرایی کد را برای کاربر نشان می دهد. کلاس ها و توابع برنامه درون این کلاس استفاده شده اند که کاربر با وارد کردن ورودی ها در هر مرحله این کلاس جزییات مربوطه را اجرا و نمایش می دهد.
  9. تابع main: این تابع نیز با فراخوانی کلاس ها هنگام اجرای برنامه اجرا شده و در صورت انتخاب کاربر با خاتمه برنامه اجراش متوقف می شود.

1. Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams:

1. Shows static structure of classifiers in a system
2. Diagram provides a basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from a business perspective

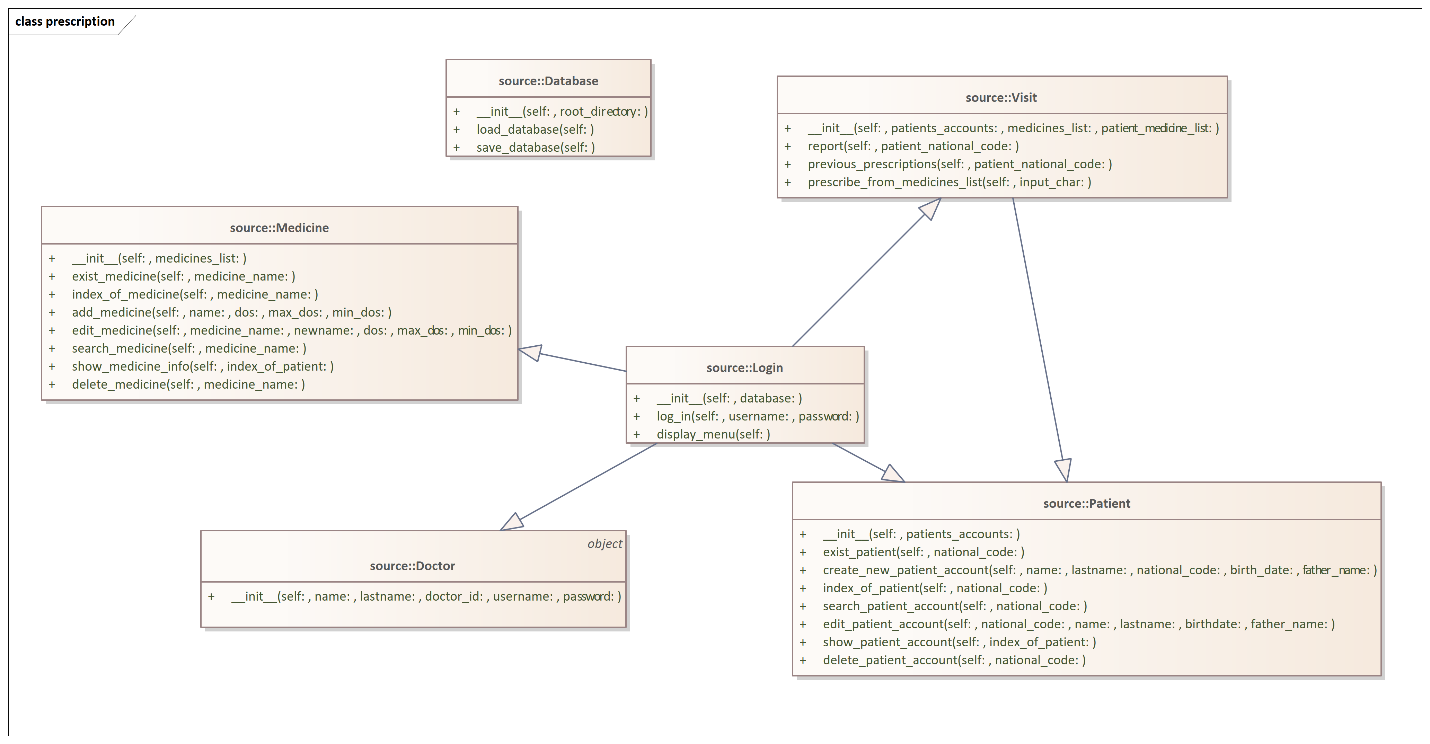


Figure 1 class diagram

1. Context Diagram:

The Context Diagram shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores, etc.).

Another name for a Context Diagram is a Context-Level Data-Flow Diagram or a Level-0 Data Flow Diagram. Since a Context Diagram is a specialized version of Data-Flow Diagram, understanding a bit about Data-Flow Diagrams can be helpful.

A Data-Flow Diagram (DFD) is a graphical visualization of the movement of data through an information system. DFDs are one of the three essential components of the structured-systems analysis and design method (SSADM). A DFD is process centric and depicts 4 main components.

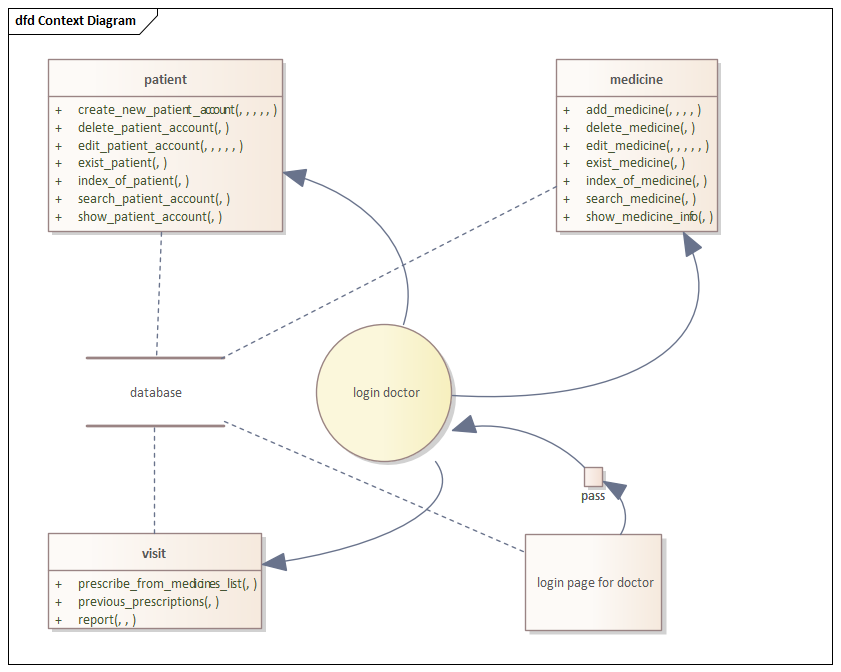


Figure 2 Context Diagram

تولید کد از روی کلاس دیاگرام  
در نرم افزار Enterprize Architecture پس از ایجاد کلاس دیاگرام از مسیر زیر و با توجه به شکل­های زیر می­توان کد مربوط به هر کلاس از کلاس دیاگرام را تولید کرد.

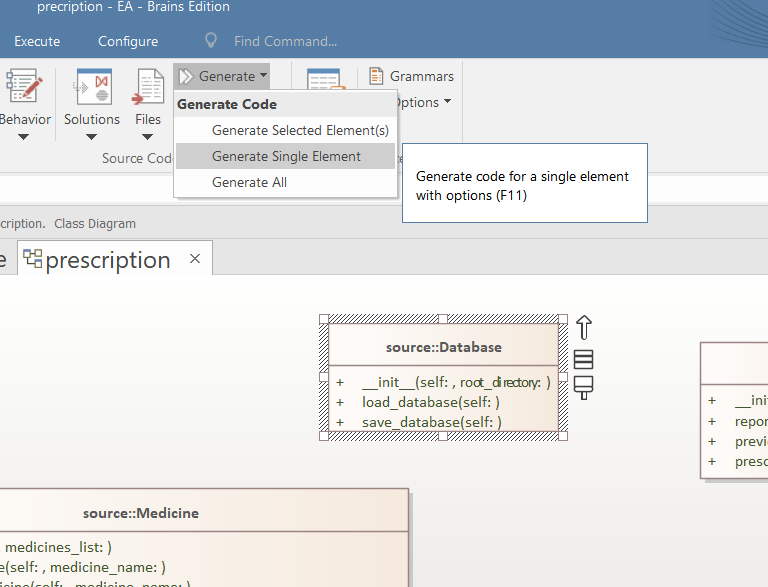
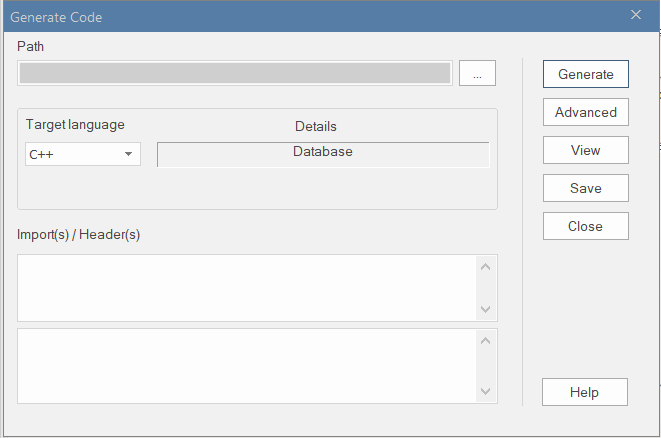
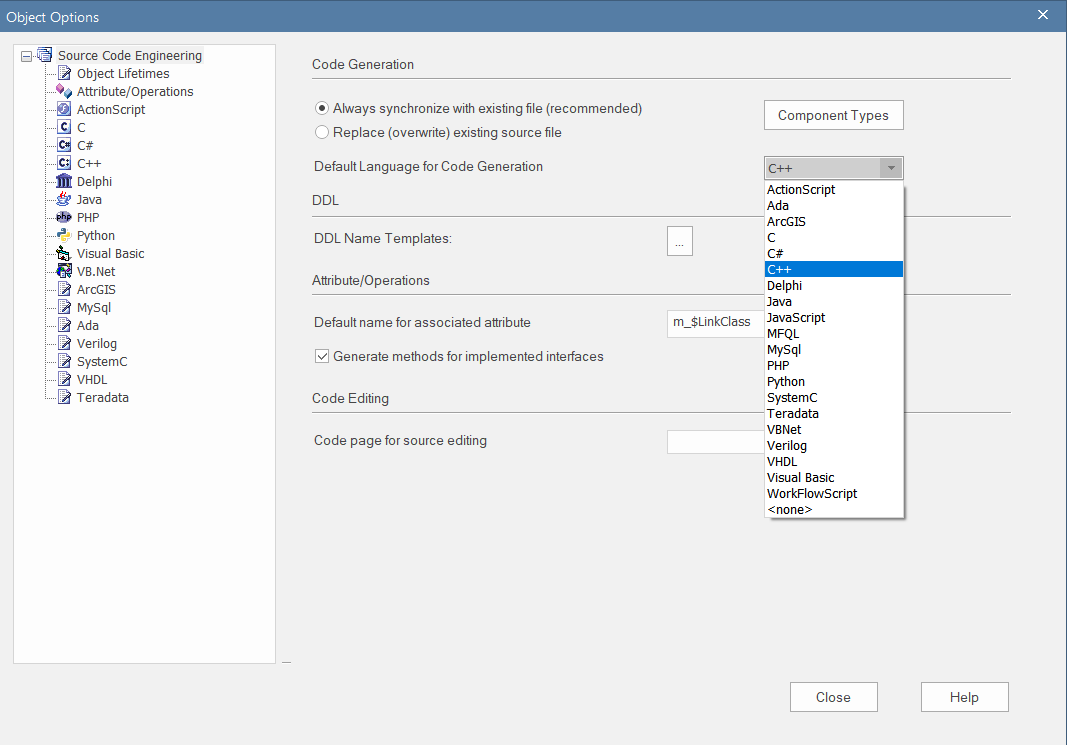


Figure 3 Generate code(1)

همان طور که در شکل بالا می بینید ابتدا یک کلاس را انتخاب سپس از تب Generte/GenerateSingle Element را انتخاب کنید.

  
Figure 4 Generate code(2)

در پنجره باز شده که در شکل بالا می بینید از سمت راست گزینه advanced را انتخاب کنید.



و در شکل بالا با انتخاب زبان مورد نظر در تب Default Lanquage For Code Generation زبان برنامه نویسی مد نظر خود برای تولید کد خروجی انتخاب کنید.

در پایان از مسیر تب Develop/Generate/Generate All می توان کد های خروجی را در دایرکتوری دلخواه تولید کرد.

## توضیحات مسیر (Directory)

از آنجایی کتابخانه های پایتون هوشمند بوده و به صورت خودکار از مسیر های مشخص معین شده در داخل کد وکتابخانه اجرا می شوند بایستی مسیر ها و پوشه های پروژه در یک مسیر و پوشه مشخص قرار بگیرند. در مثال ارایه شده فایل ها و پوشه های مورد نیاز داخل پوشه prescription ذخیره شده اند. این پوشه را در محیط برنامه نویسی باز کرده و ادامه مراحل را انجام می دهیم.

* + - 1. **TDD**: برای انجام روند تست ابتدا بایستی ماژول جدیدی برای تست ایجاد کنیم. با ایجاد فایل جدید به فرمت پایتون ابتدا نام ماژول مورد تست را وارد کنید سپس در آخر کلمه test را اضافه کرده و ذخیره کنید. ماژول تست و ماژول اصلی کد بایستی در یک مسیر ذخیره باشند.(در صورت داشتن پیکربندی های دیگر می توانید کلمه (test) را در ابتدای نام ماژول نیز بنویسید. محل این کلمه بستگی به پیکربندی ویرایشگر کد شما دارد). فایل تست را به فرمت .py ایجاد کنید.
      2. **BDD**: برای انجام روند BDD نیز لازم است اقدامات زیر انجام شوند.
    1. در مسیر فایل اصلی پوشه­ای با نام ایجاد features ایجاد کنید.
    2. در داخل پوشه فایل های فیچر با فرمت .feature ایجاد و ذخیره می شوند.
    3. پوشه steps درون پوشه featuresساخته می شود و فایل های step در این پوشه ذخیره می شوند.
    4. در داخل steps فایل های behave با فرمت .py ذخیره می شوند.
* توجه داشته باشید که برای اجرای فایل های فیچر بایستی در مسیر پوشه فیچر فایل را اجرا کنید.

برای مثال ارایه شده طبق توضیحات زیر اقدام شده است:

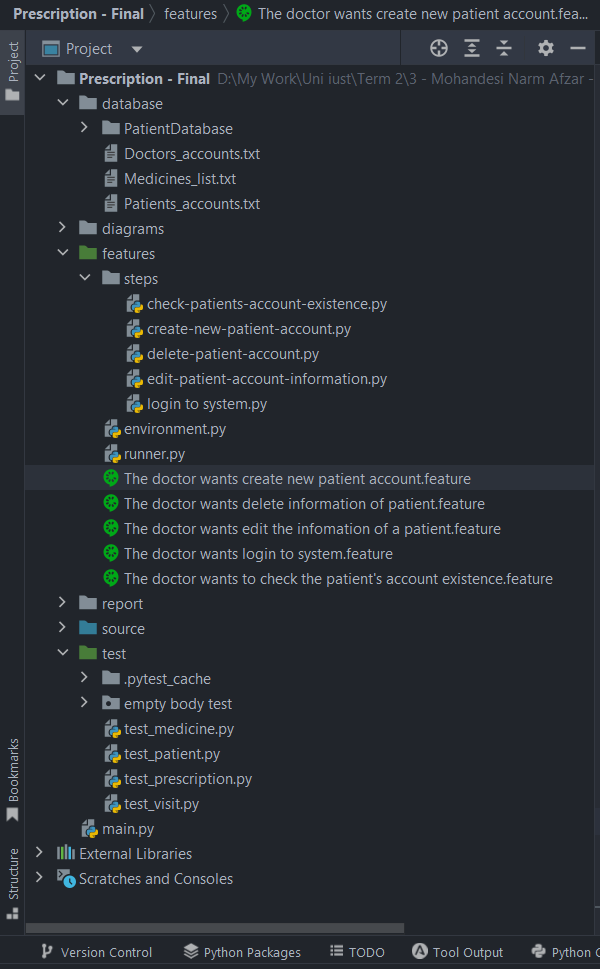


Figure 3 Directory

حال با ایجاد پوشه ها و فایل های مورد نیاز برای اجرای روند تست آماده هستید.

## توضیحات BDD

BDD is a process designed to aid the management and the delivery of software development projects by improving communication between engineers and business professionals. In so doing, BDD ensures all development projects remain focused on delivering what the business actually needs while meeting all requirements of the user.

**1.Business Goal**

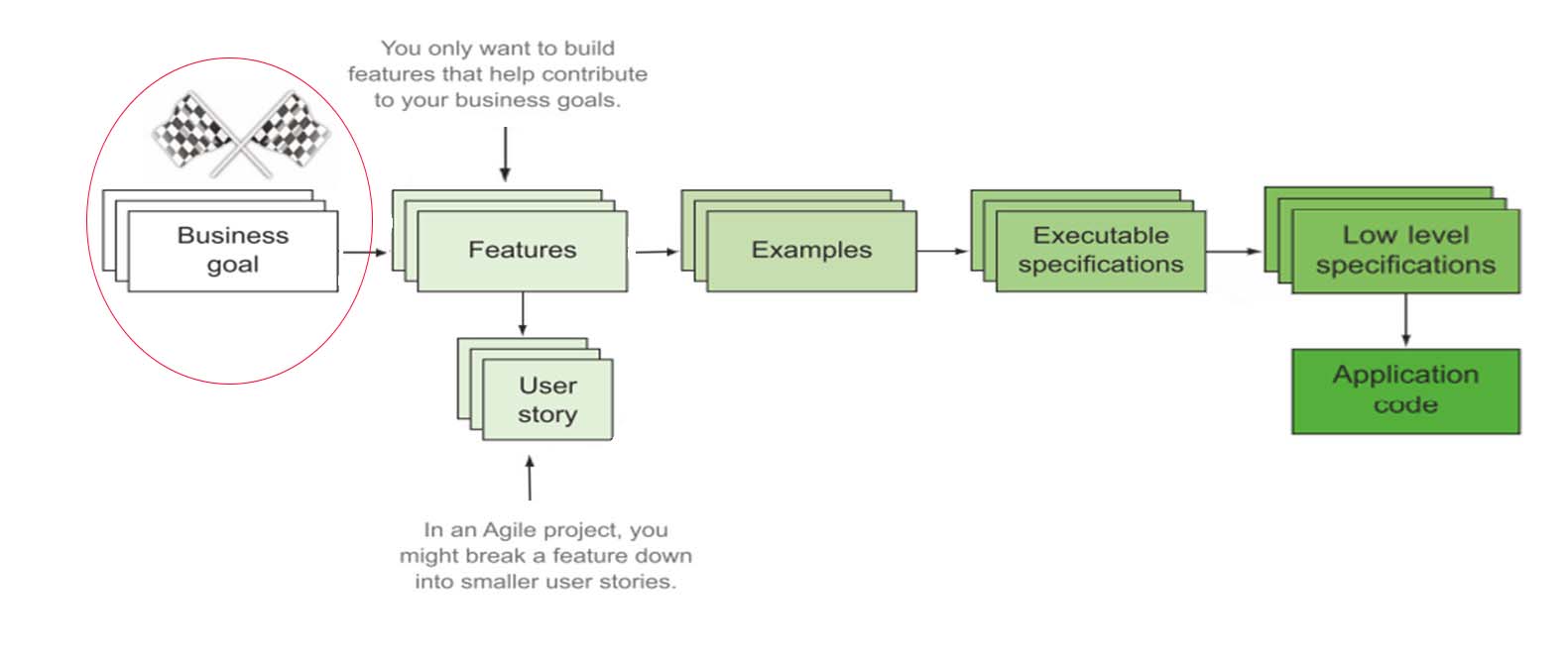


Figure 4 First Step of BDD

**Starting with a goal**

At the highest level, software development is just a way to support human behaviors by automating, replacing, or creating them. Every behavior change should help the business or the company to achieve its goals, whether that’s increasing revenues, improving usage or reducing waste.

In BDD, projects are delivered in sets of capabilities. Capabilities are essentially features in software development that enable users to be more efficient. For example, automating the ordering process of an ecommerce store or automating warehouse management.

In order to meet business requirements, you need to understand exactly what you want the software to achieve from a business perspective. In BDD, this then becomes your business goal: the driving force behind the project.

A good business goal is specific, measurable, attainable, relevant and time-bound (SMART).

Prescription business Goal:

1- Visiting and prescribing based on previous versions

2- Visiting and prescribing from the list of available drugs

At each stage after selecting the drug, the doctor determines the dose of the prescribed drug and approves the drug to be added to the final prescription.

Finally, after the drugs are added, the final prescription is prepared for the patient. Now, after the doctor's approval, it is printed and stored in the database.

**2.Features**

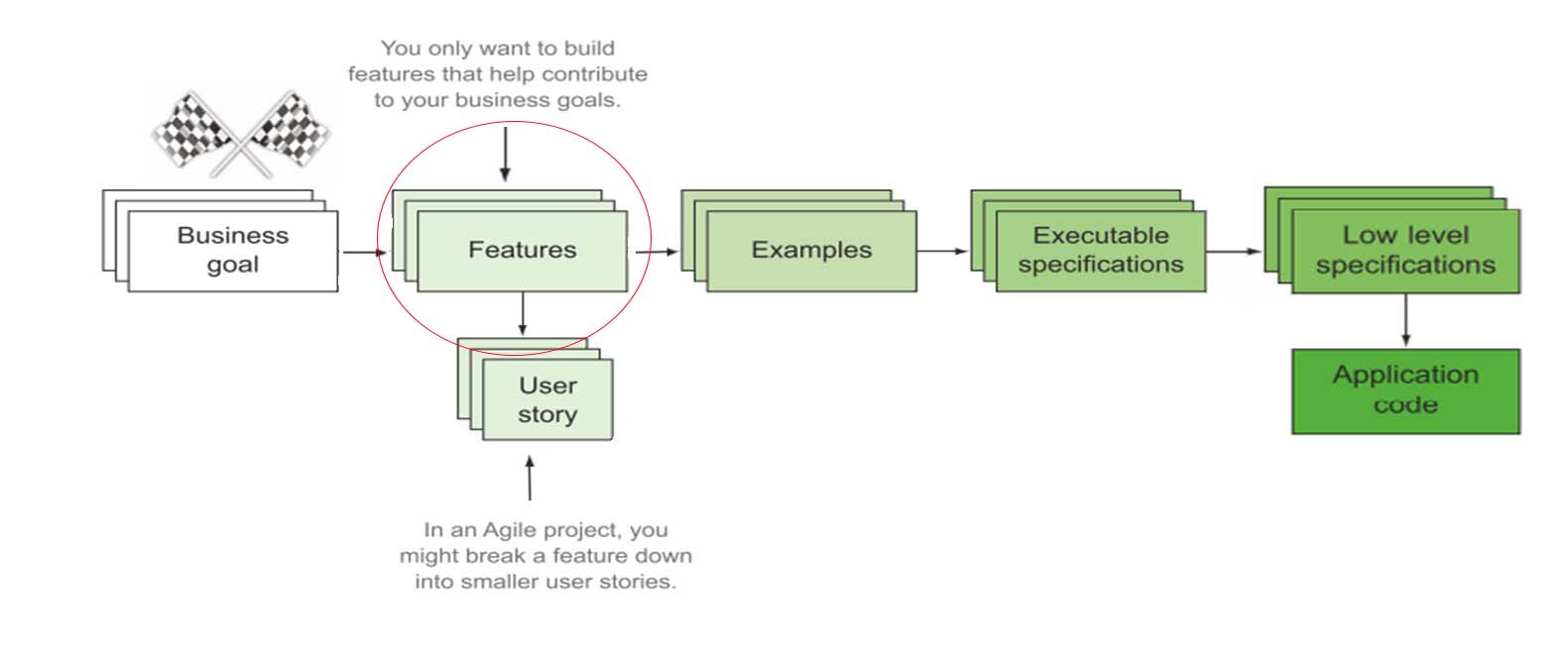


Figure 5 2nd Step of BDD

What we know from BDD is that in its core lies communication and shared understanding, and as we progress through the BDD journey we build a detailed map of the behaviors of our product in feature files. Therefore, the feature files and the information they hold play a crucial role in overall success of a project. Using the options and capabilities mentioned here truly enable you to convert feature files into lasting product documentation that would be easy to read, interactive and informative for every stakeholder.

برای انجام این مرحله ابتدا باید فایل فیچر را ایجاد کنید. در مسیر پوشه پروژه با فرمت فیچر فایلی مورد نظر را ایجاد کنید. سپس فایل را باز کرده و به الگوی زیر فیچر ها و سناریو ها را داخل فایل فیچر قرار دهید.

|  |
| --- |
| Feature: New feature  Scenario Outline:  Given  When  Then  And  Examples:  |header1 |header2 | |

Table 1 Feature file template

|  |
| --- |
| Feature: New feature  Scenario:  Given  When  Then |

Table 2 2nd template of feature file

So lets create feature file for prescription source code:

در این مثال می خواهیم تابعی که برای پس ورود و احزار هویت کاربر استفاده می شود را بصورت BDD پیاده سازی کنیم.

User story:

1.As a user I want to login

2. So I enter user and password

3. Then I login successfully

1. Feature: BDD Run for code prescription
2. As a user I want to login
3. So I enter user and password
4. Then I login successfully
6. Scenario: test\_login\_with\_out\_Examples
7. Given login to app
8. When as doctor enter: milad\_ma,1234
9. Then Login: success

پس از ایجاد فایل فیچر آن را در محیط ترمینال اجرا می کنیم. برای اجرای فایل فیچر می توان از دستورات زیر استفاده کرد توجه داشته باشید که هنگام اجرای فایل فیچر مسیر ترمینال باید با مسیر فایل فیچر یکسان باشد:

|  |
| --- |
| Cmd: ~ behave --no-capture -f plain prescription.feature  Cmd: ~ behave prescription.feature |

Table 3 command for running feature file

با وارد کردن دستور و اجرای فایل فیچر با خطای اجرا و شکست روند BDD مواجه می شویم.)توضیحات در مراحل)

**3.Examples**

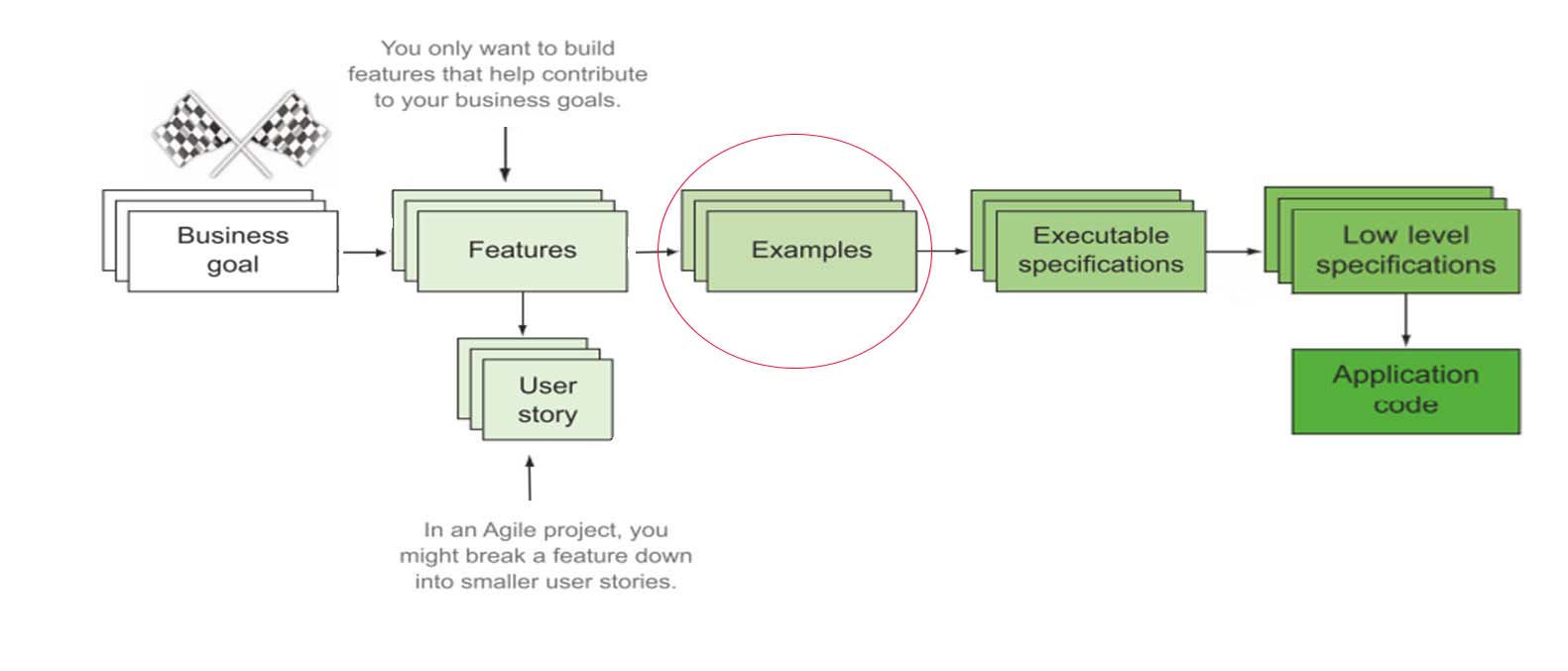


Figure 6 3rd Step of BDD

Gherkin is learned best by example. The examples cover basic, which is easy to explain and accessible to all. You can find other good example references from Cucumber and Behat. (Check the Automation Panda BDD page for the full table of contents.) [Link](https://automationpanda.com/2017/01/27/bdd-101-gherkin-by-example/)

As a disclaimer, this will focus entirely upon feature file examples and not upon automation through step definitions. Writing good Gherkin scenarios must come before implementing step definitions. Automation will be covered in future posts.

**Step Tables**

Tables are a valuable way to provide data with concise syntax. In Gherkin, a table can be passed into a step as an input. The example above can be rewritten to use a table for related results like this:

1. Feature: BDD Run for code prescription
2. As a user I want to login
3. So I enter user and password
4. Then I login successfully
5. Scenario Outline: test\_login
6. Given login for Example table
7. When as doctor types: <username>,<password>
8. Then login result <validation>
9. Examples:
10. |username           |password       |validation     |
11. |meysam\_kh          |1332           |True           |
12. |milad\_ma           |1234           |True           |
13. |hasan              |4567           |False          |
14. |hacker             |6789           |False          |

Step tables are delimited by the pipe symbol “|”. They may have as many rows or columns as desired. The first row contains column names and is not treated as input data. The table is passed into the step definition as a data structure native to the language used for automation (such as an array). Step tables may be attached to any step, but they will be connected to that step only. For good formatting, remember to indent the step table and to space the delimiters evenly.

**4.Executable Specifications:**

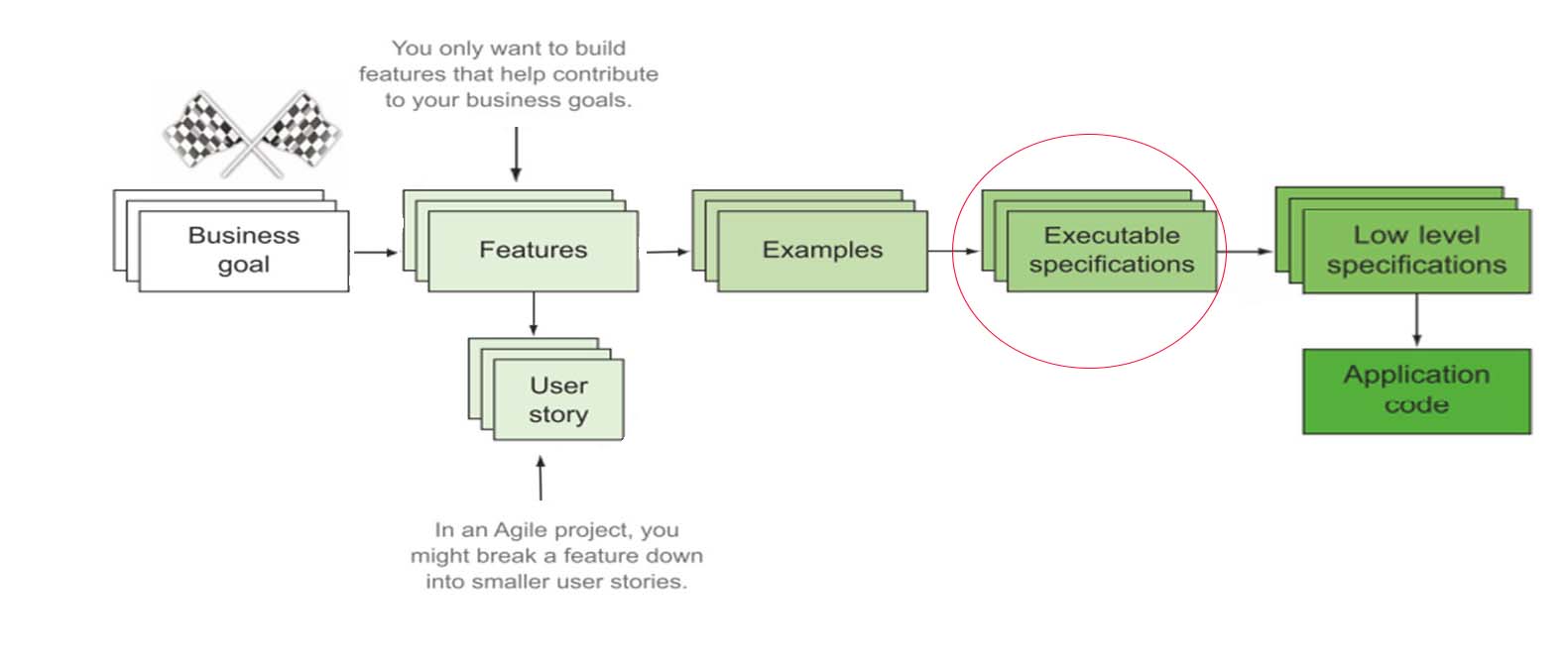
****

Figure 7 4th Step of BDD

در این مرحله باید تغییرات لازم در هر قسمت از مراحل given, when, then را تغییر داده و کد ها مورد نظر را وارد مراحل می کنیم. البته باید به این نکته توجه داشته باشید که ابتدا کتابخانه behave را فراخوانی کنید. سپس به دلیل استفاده و فراخوانی تابع log\_in از ماژول اصلی کد میز باید این کلاس و تابع را فراخوانی کنید.(به ترتیب زیر):

نمونه کد step را در شکل زیر مشاهده می کنید.

1. from behave import given, when, then
2. from package import Func
4. @given (u'')
5. def step\_impl(context):
6. print ()
7. pass
9. @when (u'')
10. def step\_impl(context):
11. print ('')
12. context. variable = Funcname()
14. @then (u'')
15. def step\_impl(context):
16. print ('')
17. assert context. variable == expected result

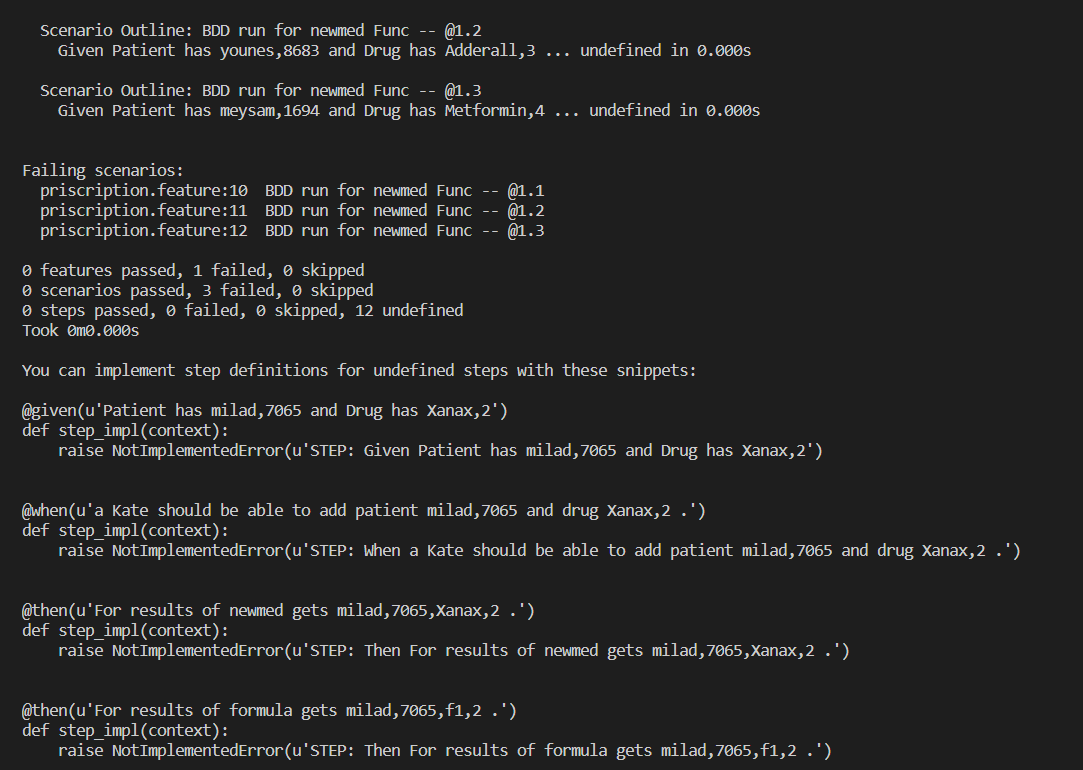


Figure 8 result of running feature file without steps

دلیل این خطا است که برای فایل فیچر فایل step به فرمت پایتون کدی نوشته نبودیم به همین دلیل در قسمت توضیحات اجرای فایل فیچر پیشنهادی برای نوشتن فایل step به ما نشان می دهد. برای ادامه پیشنهاد را کپی کرده و در فایل step جایگذاری کنید.

حال برای درک بهتر یک تابع را برای مثال انتخاب کرده و توضیحات تکمیلی را در مورد نحوه اجرا و پیاده سازی توضیح می دهیم.

با توجه به تابع ابتدا یک سناریو برای آن طراحی کنید. در قسمت scenario توضیحاتی در مورد تابعی که می خواهید تست شود اضافه شود. دقت کنید که برای هر تابع یک سناریوی جدید بنویسید همچنین

اگر می خواهید سناریویی طراحی کنید که مثال های متنوعی را تست کنید دقت کنید که از scenario Outline استفاده کنید چرا که scenario از جدول مثال ها پشتیبانی نمی کند.

پس نوشتن سناریو حال مراحل اجرای سناریو را به سرح زیر پیاده کنید:

1. در بخش Given خواسته ای که می خواهید برنامه انجام دهد را وارد کنید.
2. در بخش When شروط اجرا و نحوه اجرا و تابع اجرایی را مشخص کنید.
3. در بخش Then نیز توضیحاتی درباره نتایجی که مورد انتظار شما هستند را وارد کنید.

در نهایت در صورت نیاز جدول مثال ها را پیاده کنید. پس از طراحی و پیاده سازی سناریو ابتدا فایل فیچر را اجرا کنید همانند توضیحات بالا با خطای این که توابع در فایل step پیاده سازی نشده اند مواجه می شوید

1. from behave import given, when, then
2. from prescription import login
3. @given(u'login to app')
4. def step\_impl(context):
5. print('------- >> log in to app << -------')
6. pass
8. @when(u'as doctor enter: {user},{password}')
9. def step\_impl(context,user,password):
10. print('------- >> as doctor enter: {},{}'.format(user,password))
11. l = login()
12. context.log = l.log\_in(user,password)
14. @then(u'Login: success')
15. def step\_impl(context):
16. assert context.log == True
17. print('------- >> log in success << ------- ')

در این مرحله در ادامه فایل استپ باید مراحل اجرای سناریو را برای جدول مثال اضافه کنیم مطابق مرحله هر سه استپ را نیز این سناریو دارد پس هر سه استپ را پیاده سازی کنید توجه کنید که فرمت پیاده سازی با توجه به جدول متفاوت است.

1. from behave import given, when, then
2. from prescription import login
3. @given(u'login for Example table')
4. def step\_impl(context):
5. print('------- >> start log\_in')
6. pass
8. @when(u'as doctor types: {username},{password}')
9. def step\_impl(context,username,password):
10. print('------- >> as doctor types: {},{}'.format(username,password))
11. l = login()
12. context.log = l.log\_in(username,password)
14. @then(u'login result {validation}')
15. def step\_impl(context,validation):
16. log\_result= context.log
17. if log\_result:
18. assert context.log == True
19. print('------- >> log in success {} << ------- '.format(validation))
20. else:
21. assert context.log == False
22. print('------- >> log in fail {} << ------- '.format(validation))

**5.Low Level Specification:**

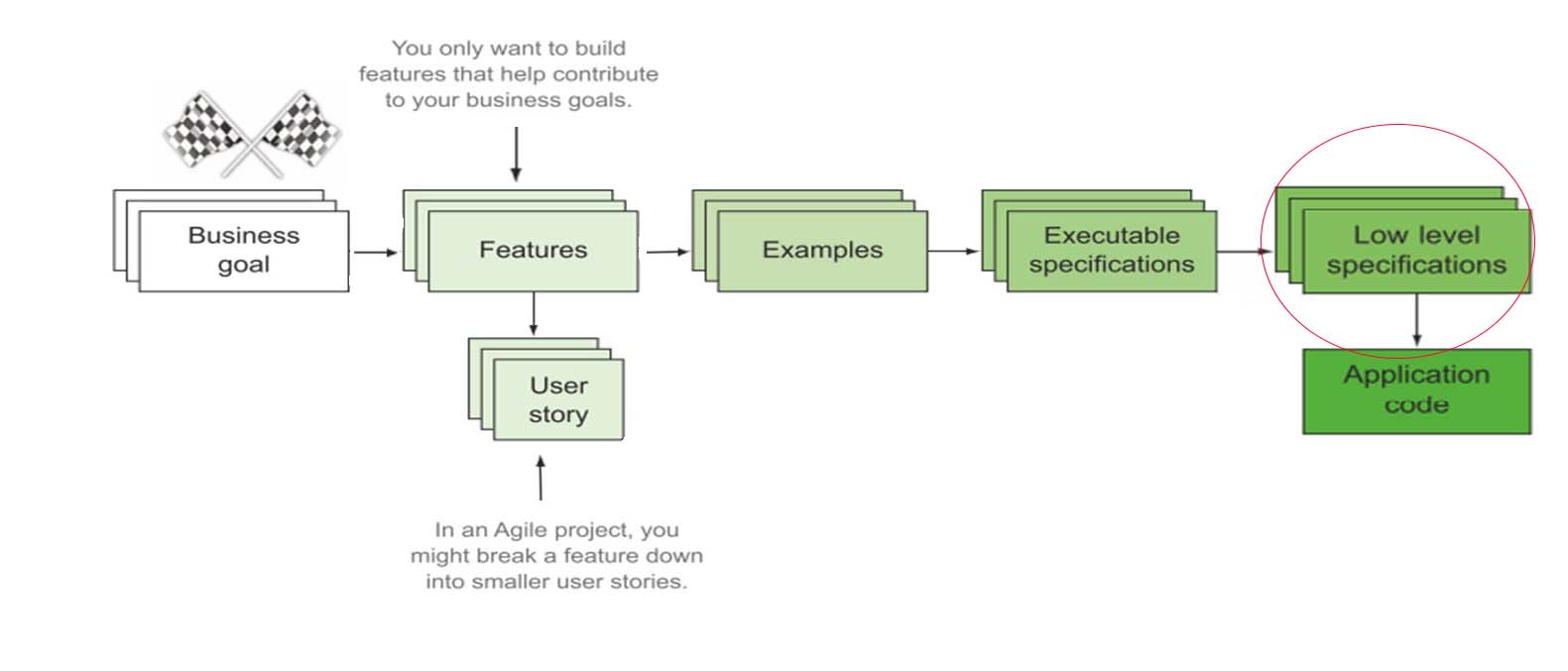


Figure 9 5th Step of BDD

پس از پیاده سازی هر سناریو بصورت فیچر و استپ حال بایستی برای هر سناریو تابع مخصوص خود را کد نویسی کنیم. از آنجایی که نیازمندی ها از قبل مشخص شده و برای هر نیازمندی یک سناریو تعریف شده است حال کافی است تنها کدی را پیاده سازی کنیم که فقط استپ های فایل فیچر با موفقیت تکیل شوند. پس در این مرحله کد های لازم را برای بدست آوردن نتیجه مورد انتظار تکمیل کرده و سپس فایل فیپر را اجرا می کنیم. در هر مرحله باتوجه به نیاز تابع بایستی تغییراتی در هر تابع داده شود. که در این مثال برای تابع های When و Then تغییرات اعمال شده است:

در تابع When : برای این مرحله تغییرات بدین شکل بوده که وقتی برنامه به مرحله login می رسد برای هر بار ورود تعدادی مثال به عنوان تست در نظر گرفته شده است بنابراین تابع از جدول مثال ها را از ستون مربوطه فراخوانی کرده و سپسدر تابع اصلی جایگذاری می کند.

در تابع Then : خروجی تابع در این مرحله ی تست مقدار مورد انتظار که در ستون اخر جدول مشخص شده را با مقدار بدست امده از اجرای تابع اصل مقایسه می کند. از آنجایی که در این تابع برای خروجی دو مقدار در نظر گرفته شده است بنابراین هر مقداری که وارد برنامه شود خروجی مورد انتظارش نمایش داده می شود.

پس از تغییرات لازم و اضافه کردن توابع لازم فایل فیچر را دوباره اجرا می کنیم. پس از اجرا نتیجه زیر در ترمینال نمایش داده می شود.

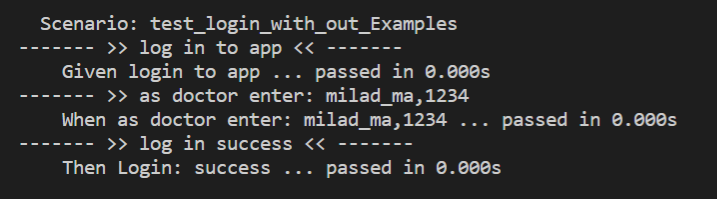


Figure 10 result of execution log\_in scenario without example

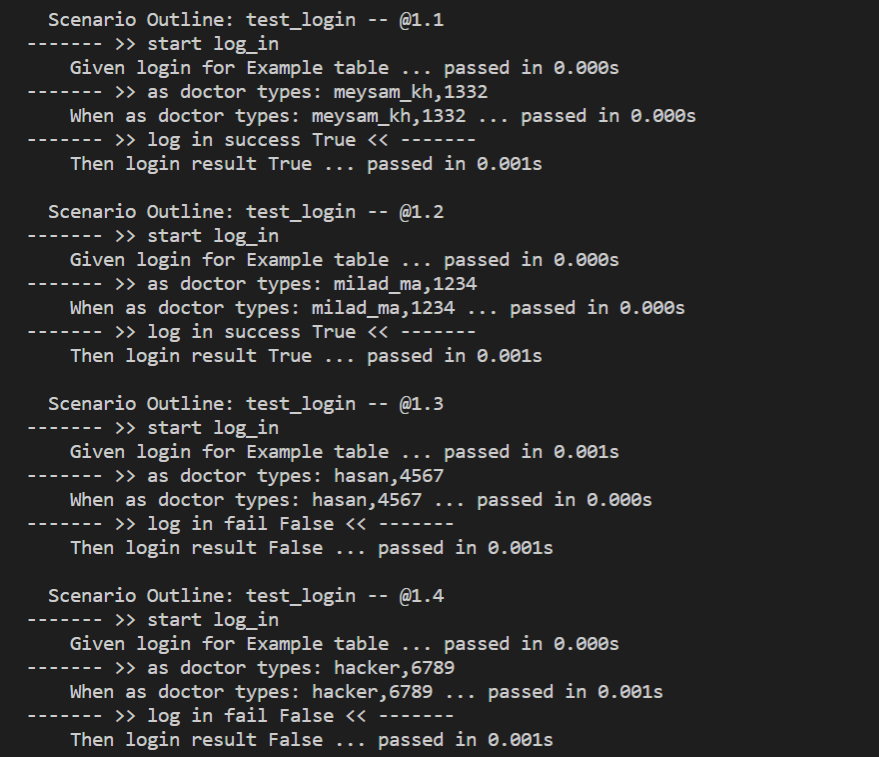


Figure 11 result of execution log\_in scenario with examples

پس از اجرا اگر پیاده سازی استپ ها به درستی صورت گرفته باشد نتایج بالا در ترمینال چاپ می شود که نشان دهنده این است که تمام مراحل و سناریو ها با موفقیت اجرا شده اند. همان طور که مشاهده می کنید برای هر مورد آزمون نتیجه اجرا را نمایش داده است. در این تابع دو مورد آخر جدول مثال ها مقدار False داشته اند که تابع به درستی اجرا شده و مقدار مورد انتظار را چاپ کرده اند.

در صورت اجرای موفقیت آمیز تمام مراحل به سراغ فایل های فیچر دیگر رفته و روند را اجرا می کنیم در غیر این صورت تغییرات لازم در کد و فایل step را عمال کرده و سپس دوباره فایل فیچر را اجرا می کنیم. این تکرار تا زمانی انجام می گیرد که تمام مراحل و سناریو ها با موفقیت به پایان رسیده باشند.

در نهایت پس از اجرای تمام مراحل نتیجه کلی به صورت نیز چاپ می شود:

1. 1 feature passed, 0 failed, 0 skipped
2. 2 scenarios passed, 0 failed, 0 skipped
3. 5 steps passed, 0 failed, 0 skipped, 0 undefined
4. Took 0m4.220s

**6.Application Code**

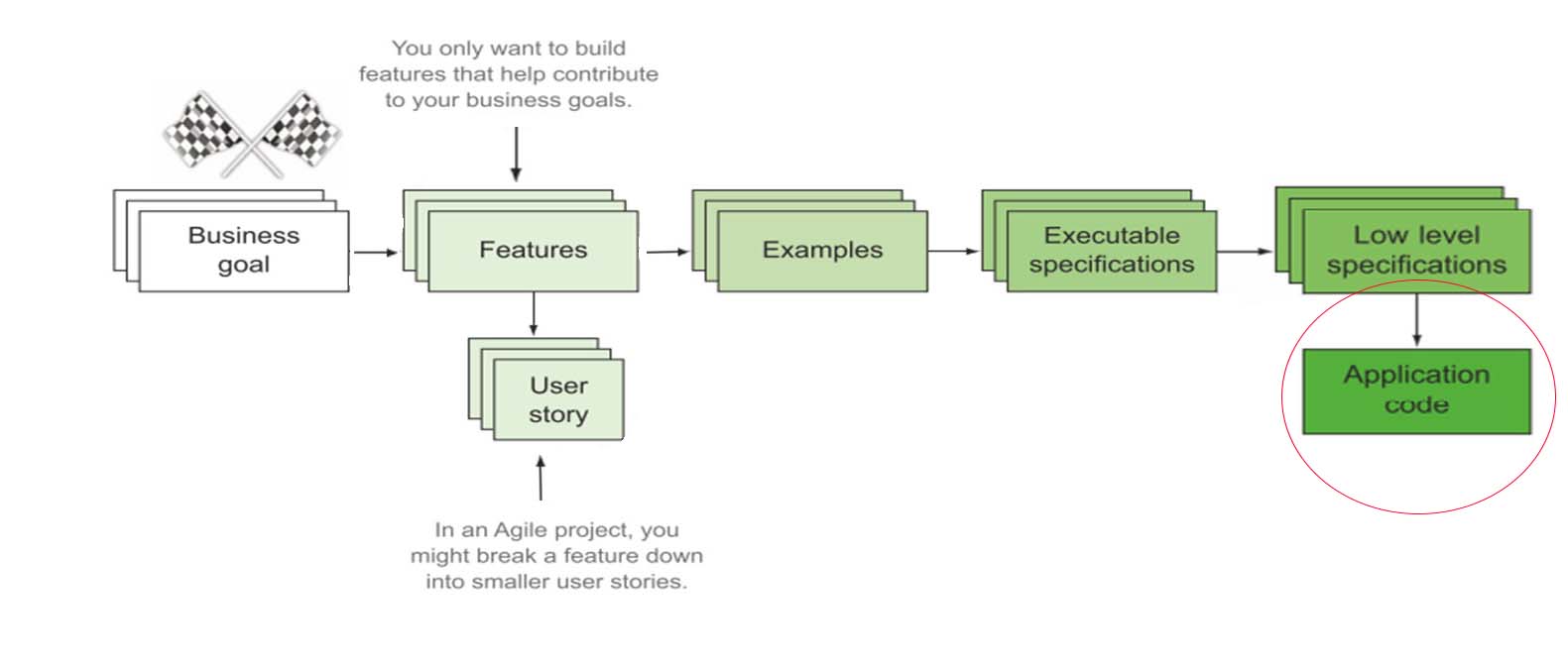


Figure 12 Final step of BDD

پس از تکمیل تمامی توابع مورد نیاز و برطرف شدن تمام نیازمندی ها و تکمیل سناریو ها حال نوبت اصلاح کد و اضافه کردن توضیحات به کد و همچنین مرتب کردن متغییر ها و دیگر موارد کد است. این مرحله به صورت یک معماری کلی دارای قواعد خاص خود است که می توانید در کتارب Clean architecture مطالعه کنید.

**برای اجرای فایل.Feature مراحل زیر را دنبال کنید:**

1- به قسمت configurations در سمت راست و بالای برنامه رفته و گزینه Edit Configurations را انتخاب کنید:

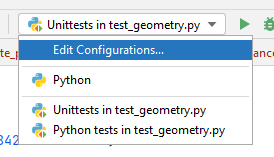


Figure 13 Add Config

2- در پنجره باز شده به قسمت سمت چپ رفته علامت + یعنی Add New Configurations را انتخاب کنید:

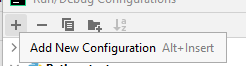


Figure 14 add new config

3- از پنجره باز شده Behave را انتخاب کنید:

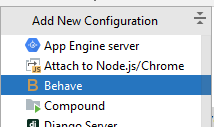


Figure 15 add behave tool

4- سپس به قسمت سمت راست پنجره رفته در قسمت Script path مسیر فایلی که می خواهید اجرا کنید را پیدا کرده و فایل را اضافه کنید:

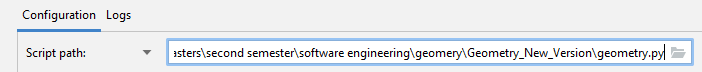


Figure 16 add source code path

5- در قسمت Python interpreter نسخه پایتون را برای اجرای فایل انتخاب کنید:



Figure 17 add interpreter for behave

6- در نهایت گزینه apply و سپس ok را انتخاب کنید.

7- در این مرحله بایستی ابزار behave را انتخاب کنید :

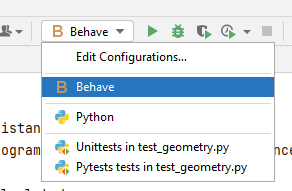


Figure 18 select behave tool

با انتخاب ابزار behave کافی است در صفحه فایل فیچر گزینه run یا کلید F5 را فشار دهید. بدین ترتیب فایل فیچر اجرا شده و نتایج در ترمینال یا صفحه خروجی سیستم شما به نمایش در می آید.

## اجرای TDD

نحوه اجرای برنامه ها :

از آنجایی که از برنامه pycharm و زبان برنامه نویسی python استفاده می کنیم لازم است که اقدامات لازم در خصوص نحوه اجرای فایل ها را بدانیم.

در ابتدا قصد داریم که فایل geometry.py را اجرا کنیم. برای تنظیمات قدم های زیر را پیش ببرید:

1- به قسمت configurations در سمت راست و بالای برنامه رفته و گزینه Edit Configurations را انتخاب کنید:

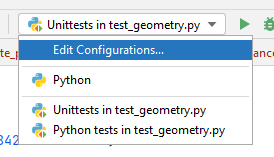


Figure 19 add config

2- در پنجره باز شده به قسمت سمت چپ رفته علامت + یعنی Add New Configurations را انتخاب کنید:

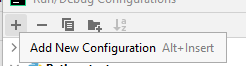


Figure 20 add configuration

3- از پنجره باز شده python را انتخاب کنید:

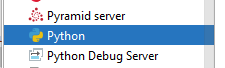


Figure 21 select language

4- سپس به قسمت سمت راست پنجره رفته در قسمت Script path مسیر فایلی که می خواهید اجرا کنید را پیدا کرده و فایل را اضافه کنید:

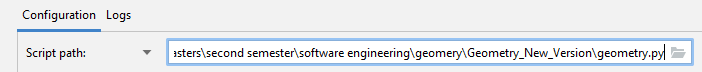


Figure 22 choose path of source code

5- در قسمت Python interpreter نسخه پایتون را برای اجرای فایل انتخاب کنید:



Figure 23 select interpreter

6- در نهایت گزینه apply و سپس ok را انتخاب کنید.

7- اکنون فقط لازم است که گزینه run را زده تا فایل اجرا شود.

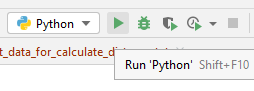


Figure 24 Run Test

در مرحله بعد برای اجرای فایل test\_geometry.py نیز مراحل زیر را پیش ببرید.

1- به قسمت configurations در سمت راست و بالای برنامه رفته و گزینه Edit Configurations را انتخاب کنید:

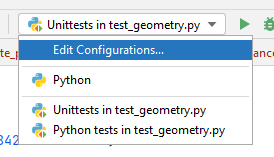


Figure 25 new config

2- در پنجره باز شده به قسمت سمت چپ رفته علامت + یعنی Add New Configurations را انتخاب کنید:

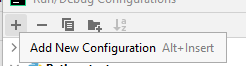


Figure 26 add Config

3-در این مرحله چون از دو ابزار تست استفاده می کنیم باید که هر دو ابزار را به برنامه اضافه نمایید:

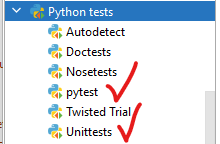


Figure 27 select testing methods

که در نهایت این ابزارها به برنامه افزوده می شوند.

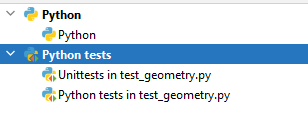


Figure 28 run with testing tool

4- سپس به قسمت سمت راست پنجره رفته در قسمت Script path مسیر فایلی که می خواهید اجرا کنید را پیدا کرده و فایل را اضافه کنید:

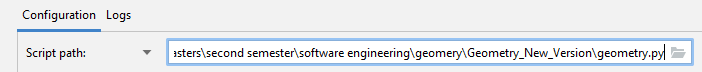


Figure 29 enter source code path

5- در قسمت Python interpreter نسخه پایتون را برای اجرای فایل انتخاب کنید:



Figure 30 select interpreter

6- در نهایت گزینه apply و سپس ok را انتخاب کنید.

7- در این مرحله باید این نکته را رعایت کنید که کدام ابزار را برای تست انتخاب نمایید :

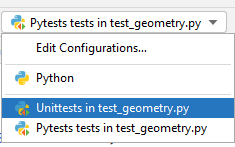


Figure 31 select testing tool

چناچنه که بخواهید که تست هایی که با unittest نوشته شده اند را اجرا کنید از ابزار unittest استفاده کنید و اگر بخواهید تست هایی که با pytest نوشته شده اند را اجرا کنید از ابزار pytest استفاده کنید.

اجرا با unittest :

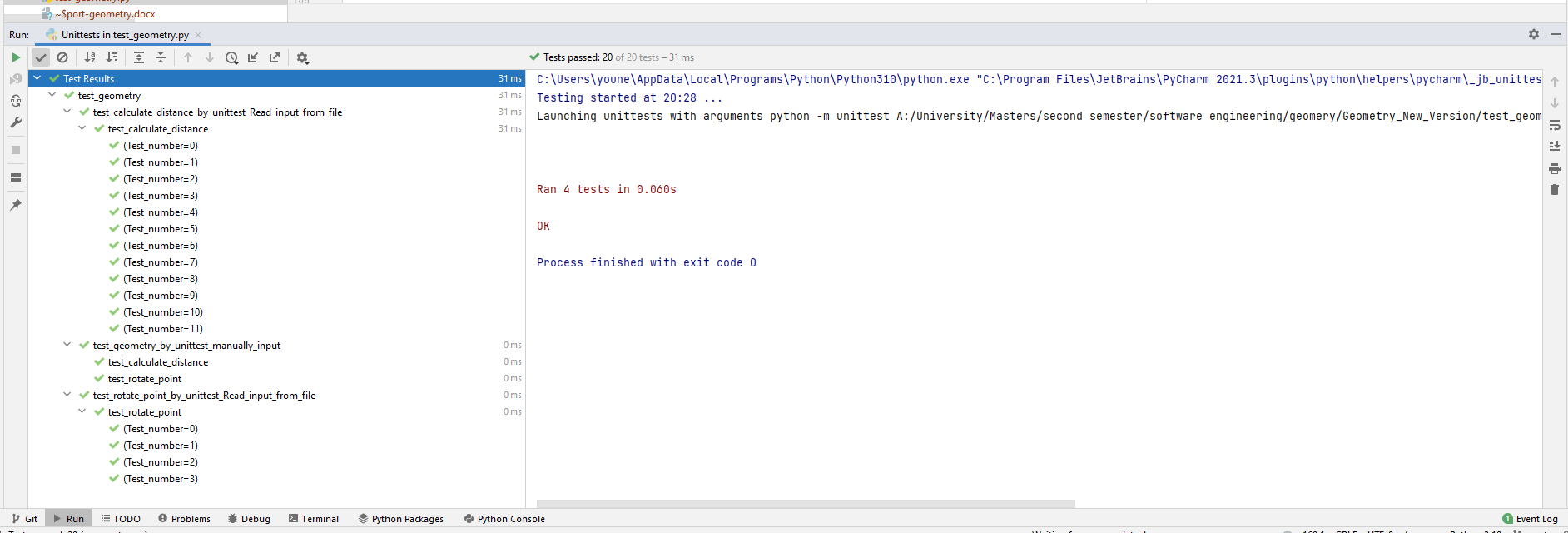


Figure 32 run with unit test

اجرا با pytest :

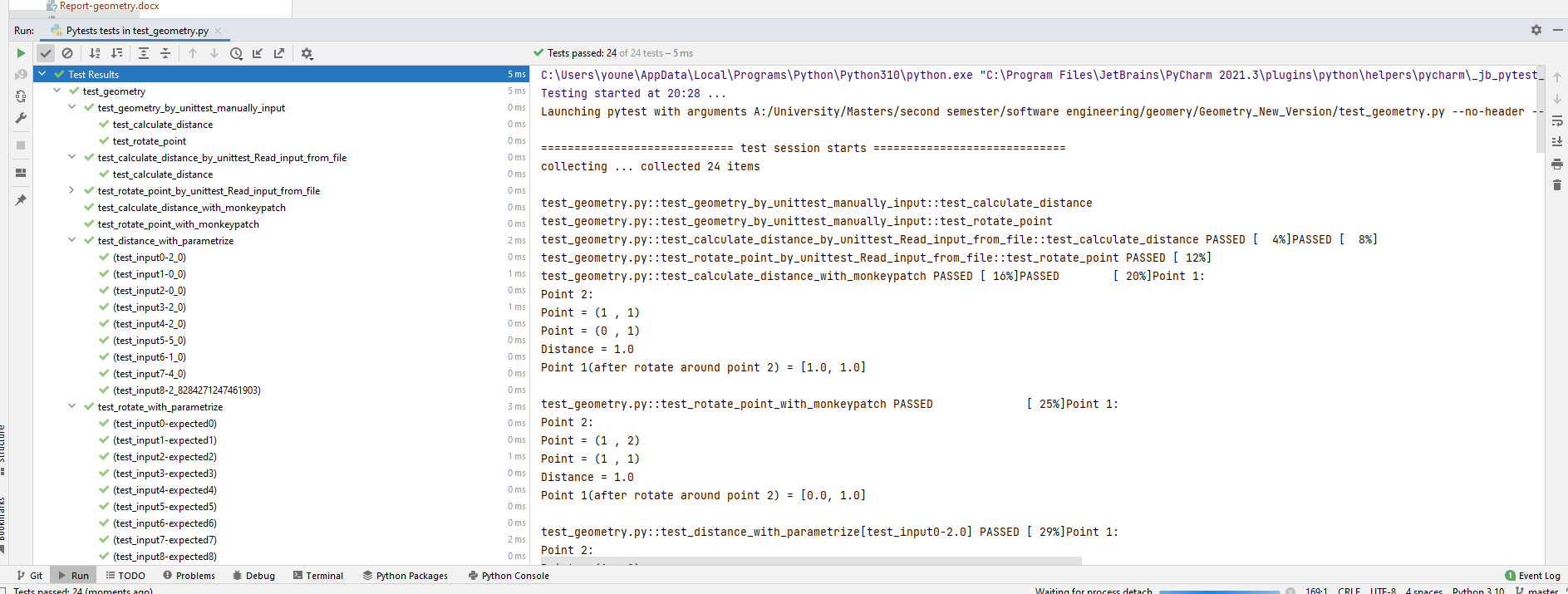


Figure 33 run with pytest

همانطور که مشاهده می کنید ابزار pytests با توضیحات بیشتری تست ها را نمایش می دهد.

حال نکته هایز اهمیت این است که pytests می تواند تست های unittest را هم اجرا کند اما در اجرای تست ها توسط pytests فقط یک دیتا را از فایل خوانده و اجرا می کند. اما unittest اصلا تست های pytest را اجرا نمی کند.

یک روش دیگر برای اجرای تست ها استفاده از ترمینال و کد دستوری است بدین ترتیب ابتدا به ترمینال رفته عبارت زیر را تایپ کنید:

1. :~ pytest -v .\test\_geometry.py

ابتدا برروی کلاس مورد نظر کلیک راست کرده و گزینه go to را انتخاب کنید، سپس گزینه Test را انتخاب کنید:

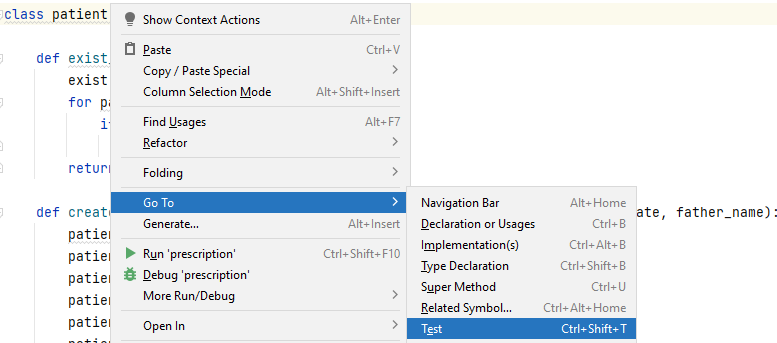


Figure 34 TDD test file creation

با انتخاب این گزینه یک پنجره برای ساخت تست ایجاد می شود:

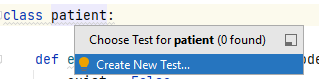


Figure 35 TDD test file creation 2nd step

در فایل تست ایجاد شده ابتدا باید کتابخانه مورد نیاز وارد شوند. پس ابتدا کتابخانه Unittest سپس توابع و کلاس های مورد تست را وارد کنید.

برای تست کد از طریق Unittest ابتدا باید کلاسی برای تست کد ایجاد کنید. کلاس تست ایجاد شده بایستی طبق الگوی زیر باشد:

|  |
| --- |
| class Test[filename] (unittest. TestCase): |

Table 4 class code for start testing

**Pre-conditions and Post-Conditions:**

تابع جدیدی به نام setUp ایجاد می کنیم. وظیفه این تابع گرفتن ورودی برای تابع اصلی است. با استفاده از این تابع برای هر تابعی که نیاز به ورودی داشته باشد قبل از اجرای تابع ورودی تنظیم می کنید. تابع دیگری که میتوان در این مرحله استفاده کرد تابع tearDown است که عمل عکس setUp را انجام می دهد. تابع باید طبق الگوی زیر ایجاد شود (حساس به حروف بزرگ و کوچک):

|  |
| --- |
| def setUp(self) -> None:  pass  def tearDown(self) -> None:  pass |

Table 5 Funcs for setUp and tearDown

حال پس از تعیین ورودی ها می توان هر کلاس یا تابعی که در بدنه اصلی کد وجود دارد را تست کنید. برای تست کافی است تابع تست برای آن کلاس یا تابع تحت تست ایجاد کنید. تابع ایجاد شده باید طبق الگوی زیر باشد در غیر اینصورت تابع به عنوان تابع تست شناسایی نمی شود.

|  |
| --- |
| def test\_ {class or function name} (self):  print("comment")  self. assertEqual ({entered input}, {expected result}) |

Table 6 Func for testing main module Funcs

در آخر نیز با وارد کردن تابع های تست می توانید به روش های زیر تست را اجرا کنید:

* 1. اگر مایل به اجرای خود تس بدون توضیحات اضافه و فقط دیدن نتایج هستید کد زیر را کد تست اضافه کرده و کد را اجرا کنید:

1. if \_\_name\_\_ == '\_\_main\_\_':
2. unittest. main ()
   1. برای اجرای تست با استفاده از افزونه ها در تب افزونه ها بخش تست را باز کرده و دکمه اجرای تست را بزنید. در صورت اجرای موفق آمیز همه تیک ها سبز می شوند در غیر اینصورت خطای تست را در بدنه تست نشان می دهد.
   2. برای اجرای دستوری و دیدن توضیحت کامل تست دستور زیر را در محیط cmd وارد کنید.

|  |
| --- |
| Cmd: ~ pytest -v priscription\_test.py |

Table 7 command for start pytest

Red Phase of test

قبل از هر کاری ابتدا باید توابع خالی در بدنه کد ایجاد کنید. سپس در ماژول تست تابعی برای تست تابع مورد نظر ایجاد کنید.

تابع تست را ایجاد کنید. با اجرا کردن تابع تست نتیجه زیر شکست خورده و با صحنه زیر مواجه می شوید:

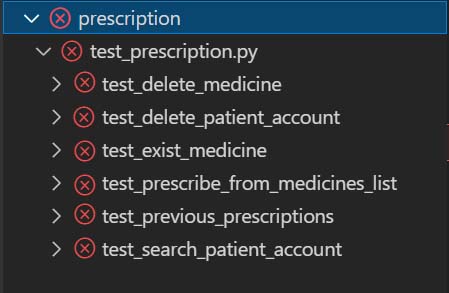


Figure 36 red phase result

دلیل شکست توابع تست، خالی بودن توابع کد برنامه است. حال به سراغ کد برنامه رفته و موارد لازم را به کد اضافه کنید. سپس تست را دوباره اجرا کنید. این تکرار تا زمانی که تمام توابع کد برنامه تشکیل شده و توابع تست آن ها نیز با موفقیت اجرا شوند ادامه می یابد.

برای مثال کد زیر را بررسی کنید:

1. from unittest import TestCase
2. class Testpatient(TestCase):
3. def test\_exist\_patient(self):
4. self.fail()
5. class Testmedicine(TestCase):
6. def test\_exist\_medicine(self):
7. self.fail()
9. class Testvisit(TestCase):
10. def test\_report(self):
11. self.fail()
12. class Testdoctor(TestCase):
13. pass
14. class Testlogin(TestCase):
15. def test\_log\_in(self):
16. self.fail()

با توجه به تعریفی که از این مرحله شد در ابتدا بدنه خالی هر تابع تست را ایجاد کنید. با اجرا کردن این کد متوجه می شوید که تمامی مراحل با شکست مواجه شده اند. حال باید با توجه به توابع تست به کد اصلی برناه رفته و با حداقل کد ممکن تابع مورد نیاز را ایجاد کنید این تابع باید به گونه ای باشد که فقط باعث موفقیت تابع تست شود.

**Green Phase of test:**

پس از اتمام روند تولید توابع تست کد برنامه تست را اجرا کنید. می بینید که تمامی مراحل با موفقیت اجرا شده و تکمیل می شوند. در این حالت پروژه به مرحله فاز سبز می رسد. در این مرحله کد برنامه شما باید تمام توابع مورد نیاز را داشته و تمام نیاز مندی ها بر طرف شده باشند. در این صورت است که با اجرای توابع تست تمامی توابع با نتیجه موفق به پایان رسیده و تیک های سبز در بخش تست ویرایشگر نمایان می شوند.

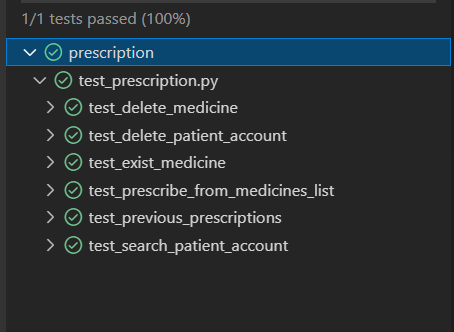


Figure 37 Green Phase result

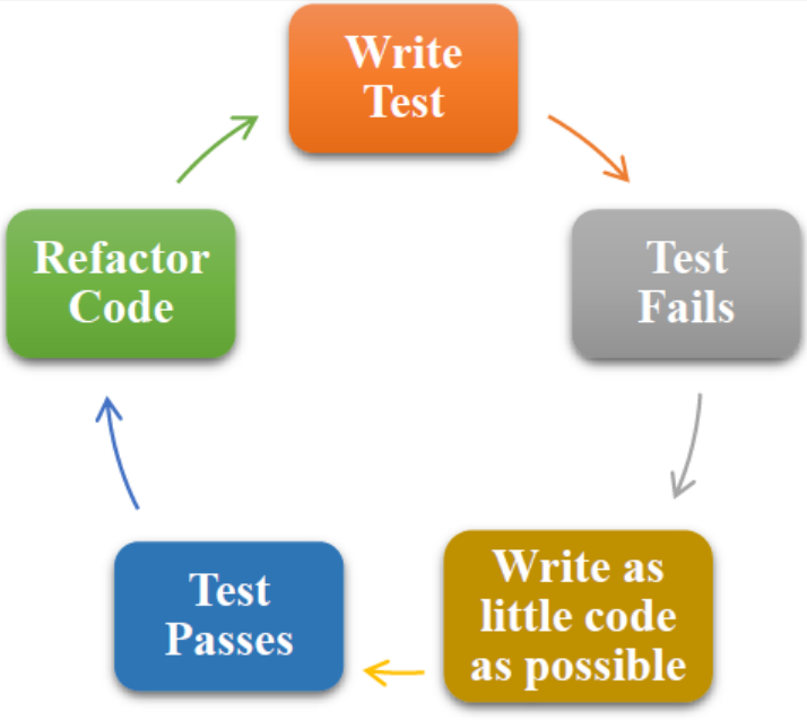


Figure 38 TDD Progress view

شکل بالا روند اجرایی TDD را نشان می دهد. پروژه در مرحله باید ابتدا به صورت تست کد شده و سپس کد تست اجرا شود که با شکست مواجه می شود. پس از این مرحله برای هر تابع تست تابع اصلی را پیاده کرده و سپس تست را اجرا کنید. در صورتی این روند به پایان می رسد که تمامی نیاز های برنامه مرتفع شده و تست های اجرای با موفقیت به پایان رسیده اند.

ساختار توابع برنامه تست:

1. def test\_search\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):

این تابع تست با استفاده از ک دملی اشخاص که بعنوان تست کیس است در داخل برنامه جستجو می کند که آیا تست کیس مورد نظر حساب کاربری بعنوان بیمار دارد یا خیر. اگر حساب کاربری موجود باشد تست قبول میشود و اگر نباشد تست با شکست مواجه می شود.

ما اکنون با استفاده از تابع مورد نظر می توانیم با یک دستور assert هر تعداد تست را می خواهیم تست کنیم.

در این برنامه ابزارmonkeypatch بعنوان mock در یونیت تست عمل نموده و ورودی ها بصورت خودکار به برنامه میدهد.

در داخل تابع هم مقادیر مورد استفاده مشخص میشوند که بعنوان stub هستند:

1. test\_data = [  
    (["4960011100"], True),  
    (["3860601325"], True),  
    (["4969053625"], True),  
    (["3860605783"], True),  
    (["4969604963"], True),  
    (["3872052053"], True),  
    (["3363363333"], True),  
    (["3360274210"], True)  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_search\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
     
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    search\_result = patient().search\_patient\_account(test\_input[0])  
    assert search\_result == expected

پس از پیاده سازی هر تابع، تابع جدیدی را برای تست پیاده سازی می کنیم.

1. test\_data = [  
    (["3860601325"], True),  
    (["4969053625"], True),  
    (["3860605783"], True),  
    (["4969604963"], True),  
    (["3872052053"], True),  
    (["3360274210"], True),  
    (["3363363332"], True),  
    (["3360274211"], True)  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    delete\_result = patient().delete\_patient\_account(test\_input[0])  
    assert delete\_result == expected

این تابع با استفاده از کد ملی بررسی می کند که آیا می توان حساب کاربری بیماری را حذف کرد یا خیر.

خروجی تابع تست پس از اجرا بصورت زیر است:

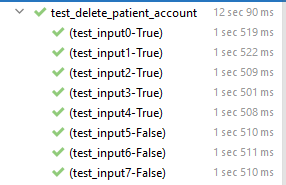


Figure 39 Func 2 green phase

تابع سوم:

1. test\_data = [  
    (["Acetaminophen"], True),  
    (["Amitriptyline"], True),  
    (["Farxiga"], True),  
    (["Invokana"], True),  
    (["Meloxicam"], True),  
    (["Naltrexone"], True),  
    (["Onpattro"], True),  
    (["Pantoprazole"], True),  
    (["Dupixent"], True),  
    (["Wellbutrin"], True),  
    (["weqwewqe"], True),  
    (["wqewq"], True),  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_exist\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    exist\_result = medicine().exist\_medicine(test\_input[0])  
    assert exist\_result == expected

این تابع تست می کند که یک دارو در لیست داروهای داروخانه موجود است یا خیر. اگر موجود باشد تست قبول می شود و در غیراینصورت تست شکست می خورد.

خروجی تابع تست پس از اجرا بصورت زیر است:

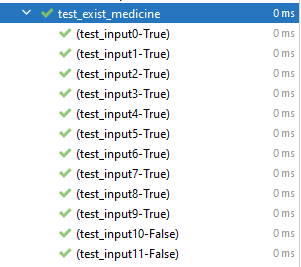


Figure 40 Func 3 green phase

تابع چهارم:

1. test\_data = [  
    (["Acetaminophen"], True),  
    (["Amitriptyline"], True),  
    (["Farxiga"], True),  
    (["Invokana"], True),  
    (["Meloxicam"], True),  
    (["Naltrexone"], True),  
    (["Onpattro"], True),  
    (["Pantoprazole"], True),  
    (["Dupixent"], True),  
    (["Wellbutrin"], True),  
    (["weqwewqe"], True),  
    (["wqewq"], True),  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    delet\_result = medicine().delete\_medicine(test\_input[0])  
    assert delet\_result == expected

این تابع نیز با استفاده از نام دارو تست می کند که می توان داروی موجود را حذف کرد یا خیر. به عبارت دیگر فقط دارو هایی را می توان حذف کرد که موجود باشند.

خروجی تابع تست پس از اجرا بصورت زیر است:

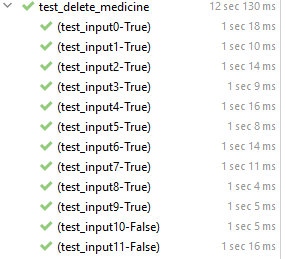


Figure 41 Func 4 green phase

تابع پنجم:

1. test\_data = [  
    (["3860960903"], ["1","1","y"], "Ozempic,1.0,1,10\n"),  
    (["3860601325"], ["3","5","y"], "Ibuprofen,5.0,1,10\n"),  
    (["3860605783"], ["1","3","y"], "Trazodone,3.0,1,10\n"),  
    (["3860960903"], ["4","4","y"], "Ativan,4.0,1,10\n"),  
    (["3872052053"], ["1","1","y"], "Lofexidine,1.0,1,10\n"),  
    (["4960011100"], ["2","2","y"], "Entresto,2.0,1,10\n"),  
   ]  
     
     
   @pytest.mark.parametrize("input\_code,test\_input,expected", test\_data)  
   def test\_previous\_prescriptions(input\_code, test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    previous\_prescriptions\_result = visit().previous\_prescriptions(input\_code[0])  
    assert previous\_prescriptions\_result == expected

این تابع با استفاده از ابزار monkeypatch ورودی ها را به صورت خودکار ذخیره کرده و سپس بررسی می کند که آیا شخصی با مقدار دز و نوع داروی مشخص شده چنین دارویی در لیست نسخه هایش هست یا خیر.

خروجی تابع تست پس از اجرا به صورت زیر است:

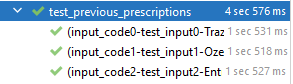


Figure 42 Func 5 green phase

تابع ششم:

1. test\_data = [  
    (["A"],["5","7","y"], "Amoxicillin,7.0,1,10\n"),  
    (["B"],["2","2","y"], "Brilinta,2.0,1,10\n"),  
    (["C"],["3","3","y"], "Citalopram,3.0,1,10\n"),  
    (["D"],["1","2","y"], "Doxycycline,2.0,1,10\n"),  
   ]  
     
     
   @pytest.mark.parametrize("input\_char,test\_input,expected", test\_data)  
   def test\_prescribe\_from\_medicines\_list(input\_char, test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    prescribe\_result = visit().prescribe\_from\_medicines\_list(input\_char[0])  
    assert prescribe\_result == expected

این تابع نیز با دادن یک حرف به عنوان ورودی، بررسی می کند که آیا دارویی با مقدار دز و مشخصات مورد نظر در لیست داروها موجود است یا خیر.

خروجی تابع تست پس از اجرا بصورت زیر است:

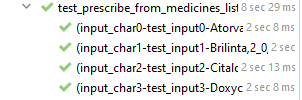


Figure 43 Func 6 green phase

با اجرای کد نهایی نتیجه زیر نمایش داده می شود.

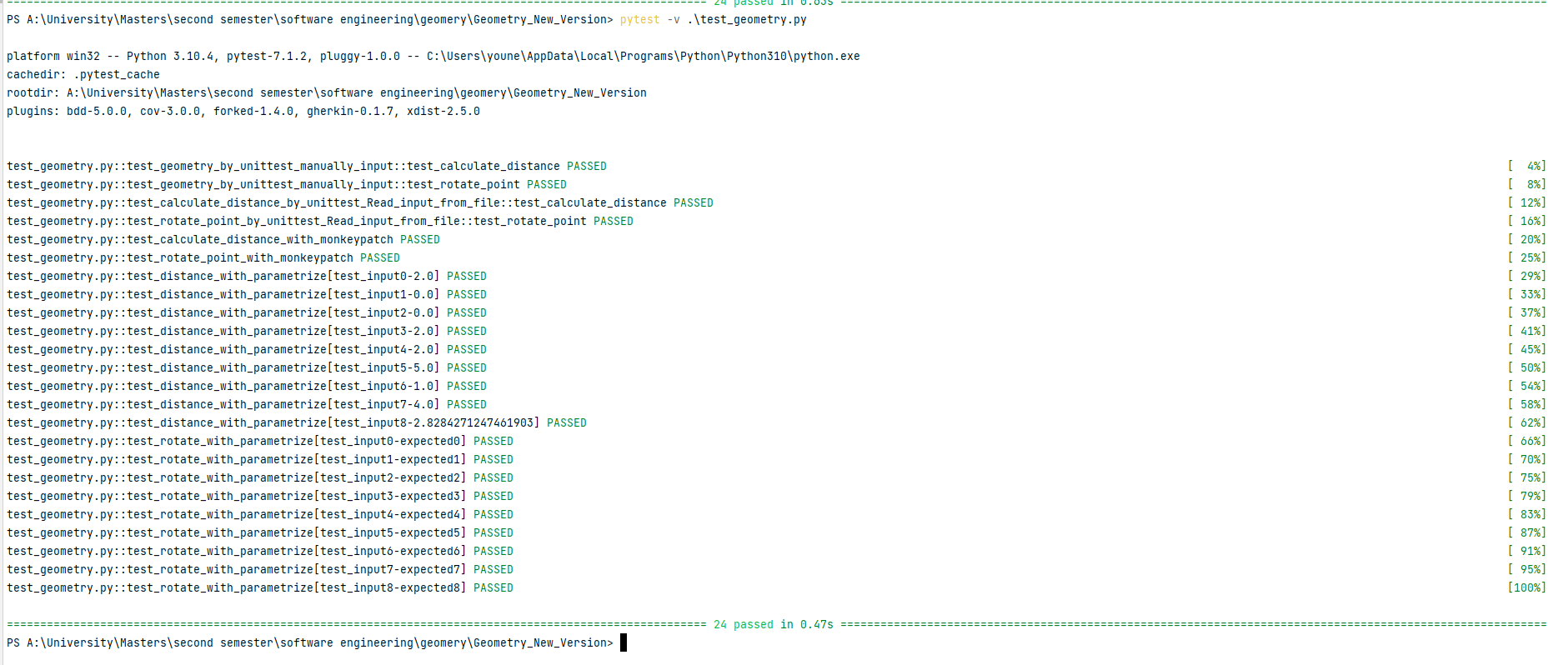


Figure 44 pytest result information

Test-Driven Development with Twisted﻿ (TDD)

Creating project, packages and Python files﻿

From the main menu, choose File | New Project... and in the New Project dialog, choose to create a pure Python project. Let's call it Calculus.

Create an empty directory called calculus containing an empty \_\_init\_\_.py file. To do that, choose File | New ... from the main menu, and then choose Python Package from the New popup menu.

Next, create a Python file. To do that, again point to File | New... from the main menu, and choose Python File in the popup. This file is called by TDD\_test\_fist\_prescription.py and it opens for editing immediately. Enter the following code:

1. from contextlib import nullcontext
2. from pathlib import Path
3. import os
4. import msvcrt
5. import time
6. import glob
7. from turtle import delay
8. from pyparsing import And
9. class load\_databas:
10. class patient:
11. def exist\_patient(self, national\_code):
12. pass
13. def create\_new\_patient\_account(self):
14. pass
15. def index\_of\_patient(self, national\_code):
16. pass
17. def search\_patient\_account(self, national\_code):
18. Pass
19. def show\_patient\_account(self, index\_of\_patient):
20. pass
21. def edit\_patient\_account(self, national\_code):
22. pass
23. def delete\_patient\_account(self, national\_code):
24. pass
25. class medicine:
26. def exist\_medicine(self, medicine\_name):
27. pass
28. def add\_medicine(self):
29. pass
30. def edit\_medicine(self, medicine\_name):
31. pass
32. def search\_medicine(self, medicine\_name):
33. pass
34. def show\_medicine\_info(self, index\_of\_patient):
35. pass
36. def delete\_medicine(self, medicine\_name):
37. pass
38. class visit():
39. def report(self, patient\_medicine\_list, patient\_natinal\_code):
40. def previous\_prescriptions(self, patient\_natinal\_code):
41. def prescribe\_from\_medicines\_list(self, input\_char):
42. class doctor(object):
43. class login:
44. class save\_databas:
45. def main():
46. if \_\_name\_\_ == '\_\_main\_\_':
47. main()

Next, right-click the package calculus and create a Python package again - this time it should be called test.

Installing packages﻿

You must install the Twisted package to execute tests. Besides that, if you are working on Windows, also install the package pypiwin32.

It's most easy to install the required packages in PyCharm: to do it, go to Settings/Preferences dialog and click the Python Interpreter page. On this page, in the Packages toolbar section, click App general add and select the required packages in the Available Packages dialog. Read the section Install, uninstall, and upgrade packages for details.

Selecting the required test runner﻿

Again, In the Settings/Preferences dialog (Ctrl+Alt+S), under the Tools node, click the Python Integrated Tools page.

On this page, select the test runner Twisted Trial from the list in the Default test runner field.

Apply changes and close the Settings/Preferences dialog.

## Creating a failing test﻿

Right-click the file TDD\_first\_test.py next to the class declaration, and press Ctrl+Shift+T or choose Go To | Test from the context menu. In the Create Test Dialog you'll have to make some changes to the default settings:

First, click the browse button next to the Target directory field, and choose test package.

Second, in the Test filename field, enter the name TDD\_test\_first\_test\_prescription.py, and in the Test class name field, enter the name. Finally, select all methods - thus the tests for all the methods of the tested class will be included.

You should get the following code:

This is a working class, but we are writing tests for Twisted... Let's import the :

1. import pytest from prescription import main, patient, medicine, visit from pytest import MonkeyPatch @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_search\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    pass @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    pass @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_exist\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    pass @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    pass @pytest.mark.parametrize("input\_code,test\_input,expected", test\_data)  
   def test\_previous\_prescriptions(input\_code, test\_input, expected, monkeypatch: MonkeyPatch):  
    pass @pytest.mark.parametrize("input\_char,test\_input,expected", test\_data)  
   def test\_prescribe\_from\_medicines\_list(input\_char, test\_input, expected, monkeypatch: MonkeyPatch):  
    pass

As the idea of TDD (Test-Driven Development) is writing tests before the code, let's make the actual tests:

1. import pytest from prescription import main, patient, medicine, visit from pytest import MonkeyPatch test\_data = [  
    (["4960011100"], True),  
    (["3860601325"], True),  
    (["4969053625"], True),  
    (["3860605783"], True),  
    (["4969604963"], True),  
    (["3872052053"], True),  
    (["3363363333"], True),  
    (["3360274210"], True)  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_search\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    search\_result = patient().search\_patient\_account(test\_input[0])  
    assert search\_result == expected test\_data = [  
    (["3860601325"], True),  
    (["4969053625"], True),  
    (["3860605783"], True),  
    (["4969604963"], True),  
    (["3872052053"], True),  
    (["3360274210"], True),  
    (["3363363332"], True),  
    (["3360274211"], True)  
   ]  
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_patient\_account(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    delete\_result = patient().delete\_patient\_account(test\_input[0])  
    assert delete\_result == expected test\_data = [  
    (["Acetaminophen"], True),  
    (["Amitriptyline"], True),  
    (["Farxiga"], True),  
    (["Invokana"], True),  
    (["Meloxicam"], True),  
    (["Naltrexone"], True),  
    (["Onpattro"], True),  
    (["Pantoprazole"], True),  
    (["Dupixent"], True),  
    (["Wellbutrin"], True),  
    (["weqwewqe"], True),  
    (["wqewq"], True),  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_exist\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    exist\_result = medicine().exist\_medicine(test\_input[0])  
    assert exist\_result == expected test\_data = [  
    (["Acetaminophen"], True),  
    (["Amitriptyline"], True),  
    (["Farxiga"], True),  
    (["Invokana"], True),  
    (["Meloxicam"], True),  
    (["Naltrexone"], True),  
    (["Onpattro"], True),  
    (["Pantoprazole"], True),  
    (["Dupixent"], True),  
    (["Wellbutrin"], True),  
    (["weqwewqe"], True),  
    (["wqewq"], True),  
   ]  
     
   @pytest.mark.parametrize("test\_input,expected", test\_data)  
   def test\_delete\_medicine(test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    delet\_result = medicine().delete\_medicine(test\_input[0])  
    assert delet\_result == expected test\_data = [  
    (["3860960903"], ["1","1","y"], "Ozempic,1.0,1,10\n"),  
    (["3860601325"], ["3","5","y"], "Ibuprofen,5.0,1,10\n"),  
    (["3860605783"], ["1","3","y"], "Trazodone,3.0,1,10\n"),  
    (["3860960903"], ["4","4","y"], "Ativan,4.0,1,10\n"),  
    (["3872052053"], ["1","1","y"], "Lofexidine,1.0,1,10\n"),  
    (["4960011100"], ["2","2","y"], "Entresto,2.0,1,10\n"),  
   ]  
   @pytest.mark.parametrize("input\_code,test\_input,expected", test\_data)  
   def test\_previous\_prescriptions(input\_code, test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    previous\_prescriptions\_result = visit().previous\_prescriptions(input\_code[0])  
    assert previous\_prescriptions\_result == expected  
     
     
   test\_data = [  
    (["A"],["5","7","y"], "Amoxicillin,7.0,1,10\n"),  
    (["B"],["2","2","y"], "Brilinta,2.0,1,10\n"),  
    (["C"],["3","3","y"], "Citalopram,3.0,1,10\n"),  
    (["D"],["1","2","y"], "Doxycycline,2.0,1,10\n"),  
   ]  
   @pytest.mark.parametrize("input\_char,test\_input,expected", test\_data)  
   def test\_prescribe\_from\_medicines\_list(input\_char, test\_input, expected, monkeypatch: MonkeyPatch):  
    """  
    :type expected: object  
    :param test\_input:  
    :param expected:  
    :param monkeypatch:  
    """ monkeypatch.setattr("builtins.input", lambda \_: test\_input.pop(0))  
    prescribe\_result = visit().prescribe\_from\_medicines\_list(input\_char[0])  
    assert prescribe\_result == expected

By the way, look at the Project tool window - it shows the structure of the your project. At the moment, you have 3 folder and 6 files:

├───1

│ TDD\_test\_first\_test\_prescription.py

│ TDD\_test\_fist\_prescription.py

│

├───2

│ TDD\_test\_first\_test\_prescription.py

│ TDD\_test\_fist\_prescription.py

│

└───3

TDD\_test\_first\_test\_prescription.py

TDD\_test\_fist\_prescription.py

**Running the failing tests﻿**

Now let's run this test. To do that, open for editing the test file TDD\_test\_first\_test\_prescription.py run the file.

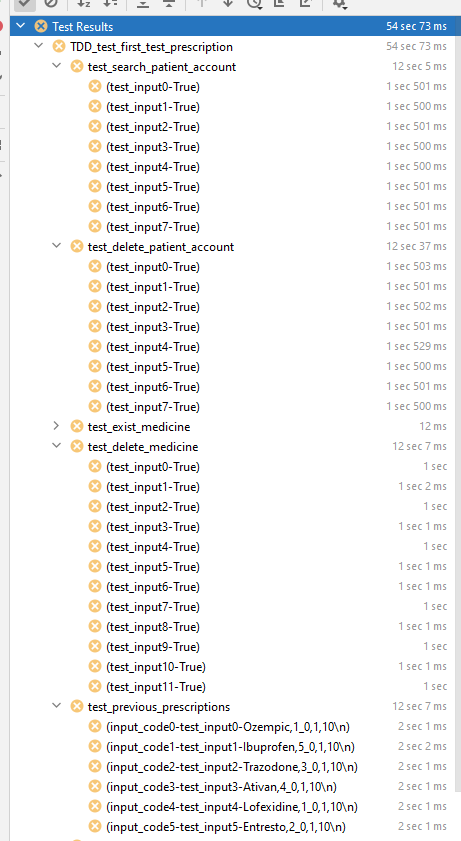


Figure 45 red phase of all tests

**Making tests pass**﻿

Let's change the class being tested and make the tests pass. To do that, we'll change the TDD\_Geometry.py script. The code of the file will look as follows:

Let's run TDD\_test\_first\_test\_prescription.py again. To do it, right-click the editor background and choose Run TDD\_test\_first\_test\_prescription.py. This time the tests pass:

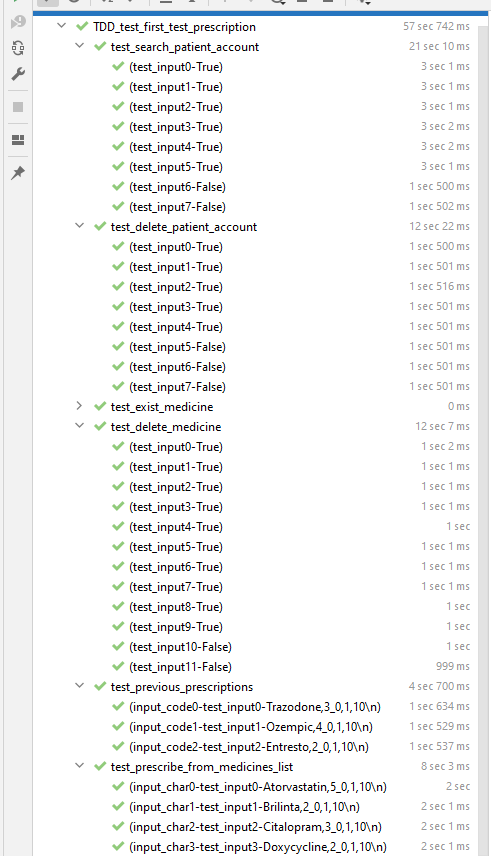


Figure 46 green phase

Figures

Figure 1 class diagram 6

Figure 2 Context Diagram 7

Figure 3 Directory 8

Figure 4 First Step of BDD 9

Figure 5 2nd Step of BDD 10

Figure 6 3rd Step of BDD 12

Figure 7 4th Step of BDD 13

Figure 8 result of running feature file without steps 14

Figure 9 5th Step of BDD 16

Figure 10 result of execution log\_in scenario without example 17

Figure 11 result of execution log\_in scenario with examples 18

Figure 12 Final step of BDD 19

Figure 13 Add Config 19

Figure 14 add new config 20

Figure 15 add behave tool 20

Figure 16 add source code path 20

Figure 17 add interpreter for behave 20

Figure 18 select behave tool 21

Figure 19 add config 21

Figure 20 add configuration 22

Figure 21 select language 22

Figure 22 choose path of source code 22

Figure 23 select interpreter 22

Figure 24 Run Test 23

Figure 25 new config 23

Figure 26 add Config 23

Figure 27 select testing methods 24

Figure 28 run with testing tool 24

Figure 29 enter source code path 24

Figure 30 select interpreter 24

Figure 31 select testing tool 25

Figure 32 run with unit test 25

Figure 33 run with pytest 26

Figure 34 TDD test file creation 27

Figure 35 TDD test file creation 2nd step 27

Figure 36 red phase result 29

Figure 37 Green Phase result 30

Figure 38 TDD Progress view 31

Figure 39 Func 2 green phase 33

Figure 40 Func 3 green phase 34

Figure 41 Func 4 green phase 35

Figure 42 Func 5 green phase 36

Figure 43 Func 6 green phase 36

Figure 44 pytest result information 37

Figure 45 red phase of all tests 42

Figure 46 green phase 43

Tables

Table 1 Feature file template 11

Table 2 2nd template of feature file 11

Table 3 command for running feature file 12

Table 4 class code for start testing 27

Table 5 Funcs for setUp and tearDown 28

Table 6 Func for testing main module Funcs 28

Table 7 command for start pytest 28

*References*

1. [*https://code.tutsplus.com/tutorials/behavior-driven-development-in-python--net-26547*](https://code.tutsplus.com/tutorials/behavior-driven-development-in-python--net-26547)
2. [*https://www.tutorialspoint.com/behave/behave\_quick\_guide.htm#*](https://www.tutorialspoint.com/behave/behave_quick_guide.htm)
3. [*https://github.com/tfio/bdd-testing-quickstart*](https://github.com/tfio/bdd-testing-quickstart)
4. [*BDD - TDD in a BDD Way (tutorialspoint.com)*](https://www.tutorialspoint.com/behavior_driven_development/tdd_in_a_bdd_way.htm)
5. [*BDD Testing Framework | PyCharm (jetbrains.com)*](https://www.jetbrains.com/help/pycharm/bdd-frameworks.html)
6. [*BDD - Specifications by Example (tutorialspoint.com)*](https://www.tutorialspoint.com/behavior_driven_development/bdd_specifications_by_example.htm)
7. [*unittest — Unit testing framework — Python 3.10.4 documentation*](https://docs.python.org/3/library/unittest.html)
8. [*Unit Testing in Python - Unittest - GeeksforGeeks*](https://www.geeksforgeeks.org/unit-testing-python-unittest/)
9. [*Getting Started With Testing in Python – Real Python*](https://realpython.com/python-testing/)
10. [*unittest.mock — mock object library — Python 3.10.4 documentation*](https://docs.python.org/3/library/unittest.mock.html#:~:text=mock%20is%20a%20library%20for,stubs%20throughout%20your%20test%20suite.)
11. [*Python Unittest Vs Pytest: Choose the Best - Python Pool*](https://www.pythonpool.com/python-unittest-vs-pytest/#:~:text=Pytest%20has%20rich%20inbuilt%20features,assert%20the%20conditions%20inside%20them.)
12. [*python - unittest Vs pytest - Stack Overflow*](https://stackoverflow.com/questions/27954702/unittest-vs-pytest)
13. [*pathlib — Object-oriented filesystem paths — Python 3.10.4 documentation*](https://docs.python.org/3/library/pathlib.html)
14. [*How to monkeypatch/mock modules and environments — pytest documentation*](https://docs.pytest.org/en/7.1.x/how-to/monkeypatch.html)
15. [*unittest — Unit testing framework — Python 3.10.4 documentation*](https://docs.python.org/3/library/unittest.html#setupclass-and-teardownclass)