**2023/11/25 00:00|Лекция. Создание исключений**

## Создание исключений

На этом уроке мы будем изучать создание исключений, как простых исключений, которые включают в себя по сути текстовое описание ошибки, так и создание более продвинутых исключений классов.

#### Для начала поговорим о raise и его функциях.

С помощью ключевого слова raise мы можем создать быстрый класс ошибки, придумать ему некоторое текстовое описание и вызвать эту ошибку на каком-либо уровне. Вспоминаем прошлые уроки: если это было вызвано где-то в функции и не обработано на этом же самом уровне функции, тогда она перебрасывается наверх и так далее. Вспоминаем пример с башенкой и то, как идет потихонечку наше исключение, пока его где-то не схватят или оно не разобьётся об консоль. По сути, это все функции у raise.

**Зачем нам вообще нужно raise?**

1.Чтобы генерировать исключения, когда функции передаются некорректные аргументы. То есть функция может работать и с некорректными аргументами, никакое исключение из-за этого вызываться потенциально не может, но если так было не запланировано разработчиком/программистом, тогда нам нужно как-то от этого избавиться. Нужно предупредить пользователя о том, что неправильно пользуются какой-либо функцией или какой-либо частью кода. Поэтому мы можем именно в этом моменте, если мы переловим эти неправильные аргументы, сравним их с чем-нибудь - вызвать исключение, назвать как-то класс и передать его на уровень повыше, где мы можем точно так же, снова перекинуть это исключение и обработать на каком-то уровне, если нам это необходимо.

2.Если определённая часть кода достигает состояния, при котором дальнейшее выполнение невозможно или не имеет смысла. То есть точка, при которой код исполнен или конец - вызываем исключение.

3.Если обнаруживается ошибка в логике вашего года, которая не должна была произойти при корректной работе. То есть если программист что-то не запланировал, тогда мы вызываем своё собственное исключение.

**Давайте рассмотрим пример(рис.1).**

Тут у нас есть функция greet\_person, которая принимает на вход только аргумент person\_name и у нас потом сравнивается, если person\_name==’ВоланДеМорт’, то мы генерируем на этом этапе исключение. Исключение у нас называется Exception. Лучше называть исключение, либо которое есть в Python, либо мы в будущем научимся создавать свои. Какие-то рандомные слова не ставим, потому что сгенерируется уже исключение о том, что не существует такого исключение, то есть исключение будет nameError, потому что такого исключения нет, значит: нет такой переменной, нет такого объекта. Так вот мы тут генерируем исключение с помощью конструкции raise, пишем название класса ошибки и потом сюда в текст передаем аргументы этой ошибки.

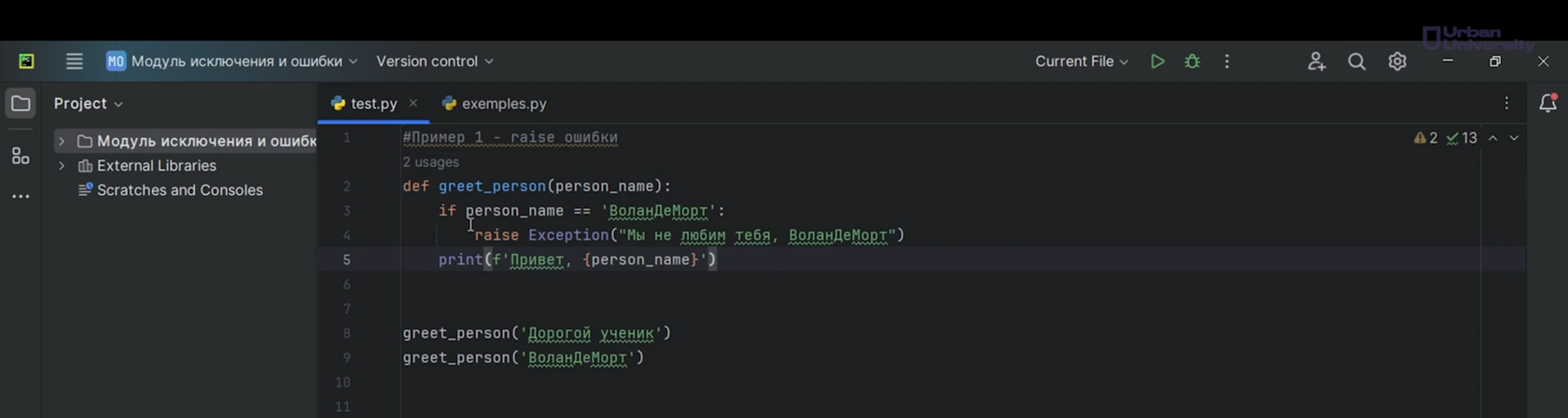


Рис.1

Как вы помните из прошлого занятия, мы можем, если перехватим нашу ошибку, сохранить класс ошибки в переменную, а с помощью параметров переменной, с помощью аргументов переменной, мы можем понять - какие аргументы есть у этой ошибки. В нашем случае аргументом будет “Мы не любим тебя, ВоланДеМорт”. Это базовая конструкция, то, как мы можем работать с raise. То есть, если у нас аргумент который мы передаем функции не равен “ВоланДеМорт”, то у нас просто выводится строчка ‘Привет, {person\_name}’. Если мы сперва передаём аргумент ‘Дорогой ученик’, выводится “Привет, Дорогой ученик”(рис.2). Как только мы передаём сюда ‘ВоланДеМорт’ - во-первых, у нас выполняется строчка “Привет, Дорогой ученик”, во-вторых, вызывается ошибка Traceback(рис.3), потому что мы сгенерировали исключение и нигде его не перехватили, что в целом логично. Exception: “Мы не любим тебя ВоланДеМорт”, то есть исключение вызвалось, мы его сделали на том уровне, который нам необходим. Мы его сами, по сути, вызвали. Конкретно здесь мы запланировали так, чтобы пользователи не вводили сюда аргумент “ВоланДеМорт”.

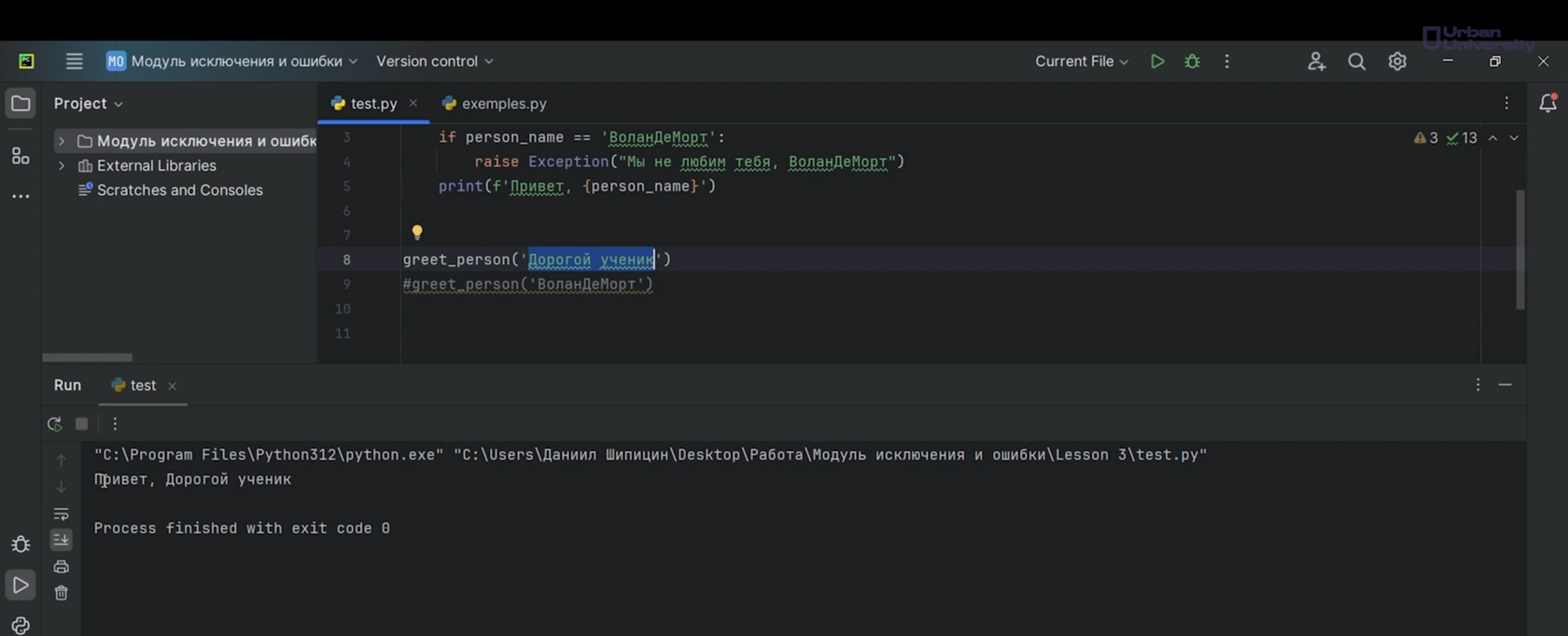


Рис.2

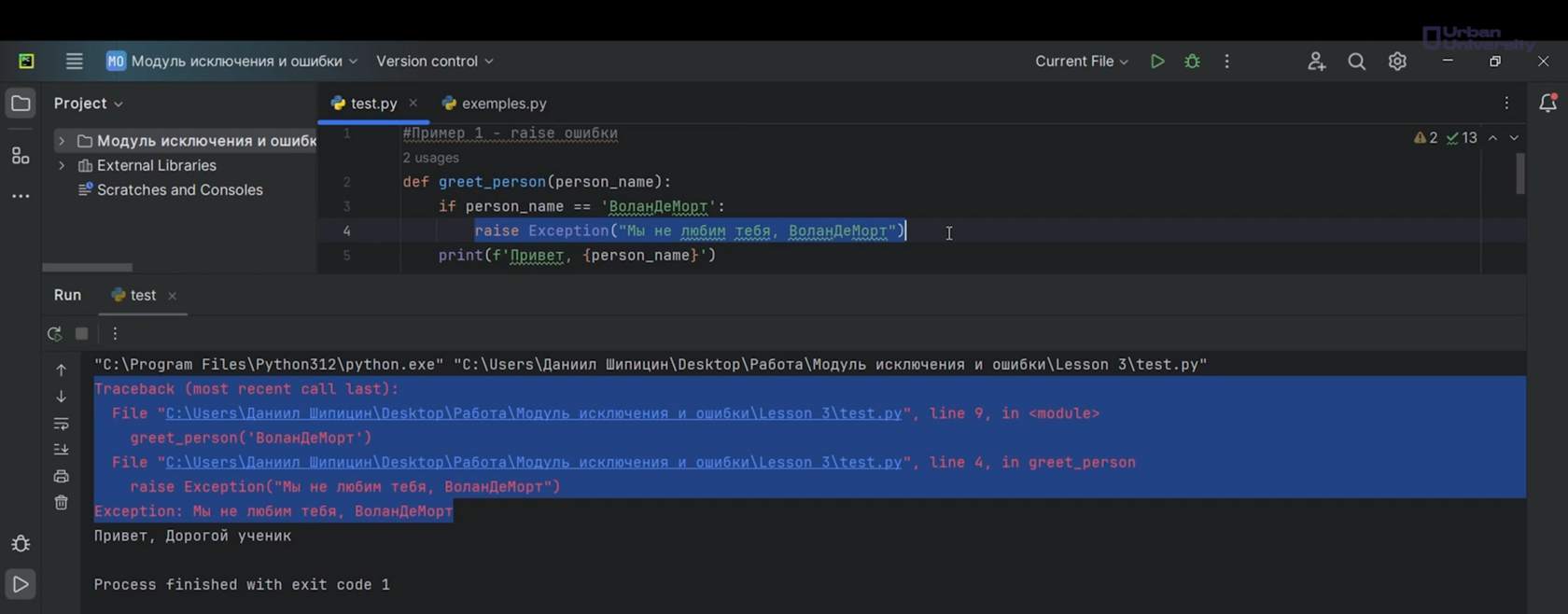


Рис.3

**Давайте рассмотрим 2 пример(рис.4).**

Здесь мы точно так же на 3 строчке генерируем ошибку, но мы её сразу же перехватываем с помощью блока try и except. Так как мы генерируем ошибку, NameError(‘Привет Там’), то мы перелавливаем ошибку NameError, сохраняем её в переменной, потом вводим эту переменную, выводим тип этой ошибки и выводим аргументы этой ошибки. Помимо этого, мы её перебрасываем дальше, то есть, как только мы перехватили эту ошибку, у нас она перехватилась на этом уровне, но мы дальше генерируем исключение, то есть raise, без какого-либо параметра, мы просто перебрасываем ошибку дальше наверх.

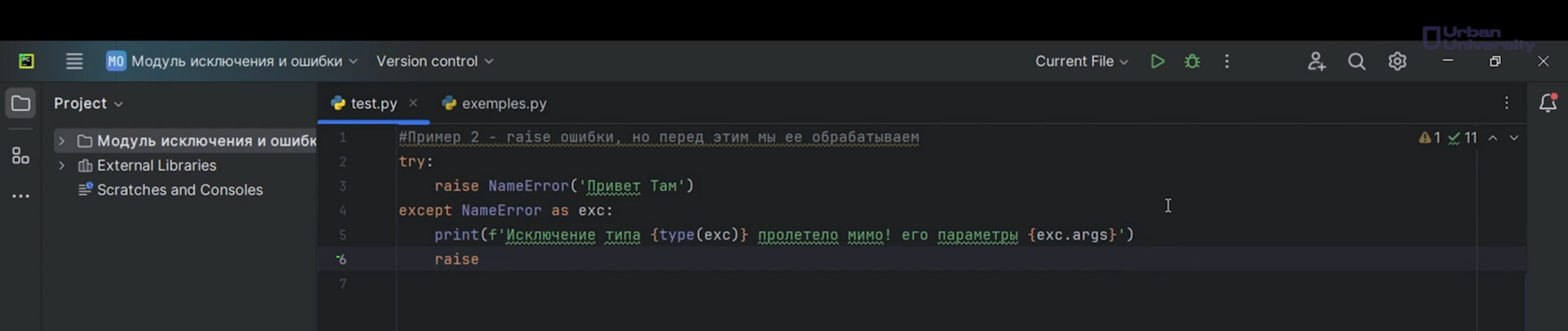


Рис.4

Выходит следующее(рис.5): Исключение типа (так мы указали type, у нас выводится тип нашего объекта) пролетело мимо! Его параметры (‘Привет Там’)”, потому что мы указали “Привет Там", оно действительно пролетело мимо, мы как бы его обработали, но из-за этого блока raise, который идёт уже после except, точнее внутри except, оно перебрасывается наверх, потому что генерируется новым исключением и снова у нас появляется TraceBack, потому что raise с 6 строки мы уже нигде не обрабатываем. Нам это необходимо так как ошибки лучше обрабатывать именно на том уровне, который задумал программист.

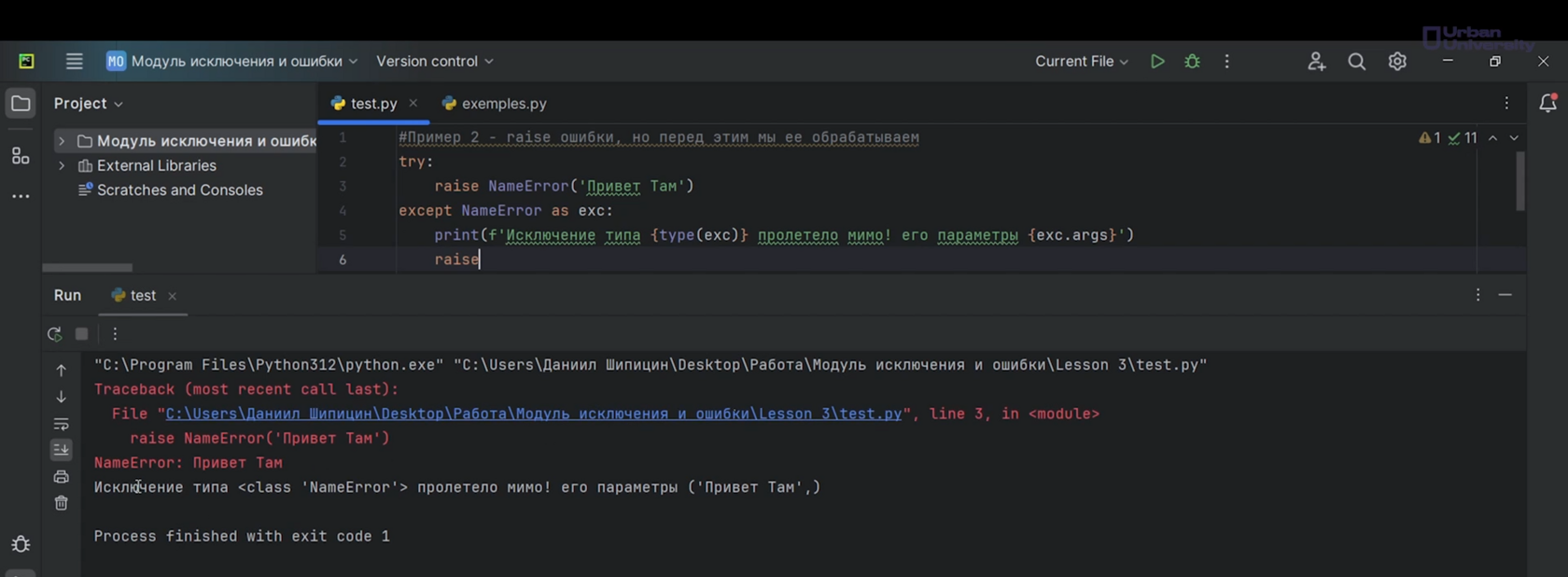


Рис.5

Если у вас многослойная программа, очень много функций, внутри функций вызывается, 1000 классов наследуется друг от друга, то лучше посвятить какому-то определённому классу именно обрабатывание ошибок, обрабатывание всех исключений, а не засовывать это всё в один класс.

Часто такое бывает, что нужно сделать собственный класс ошибок или класс ошибок, который мы по-своему назовем, они будут работать по-своему, то есть по-своему выдавать информацию об исключительной ситуации. Это надо, когда мы, например, пишем сбой собственного framework, библиотеку с нуля, тогда, когда может возникнуть много исключительных ситуаций именно с тем, как работает пользователи, а вшитых внутрь python нашивок не хватает. Вот именно тогда мы приступаем к созданию собственных ошибок.

#### Как сделать свой собственный класс ошибок?

1.Для начала нам нужно придумать класс, назвать его правильно и наследовать его от класса Exception, можно и от других классов, но очень удобно именно наследовать от класса Exception, потому что он и предназначен как раз для того, чтобы работать с исключениями, которые нужно перехватывать.

2.Можно создать продвинутый класс, с более подробным описанием, со своими функциями, методами, характеристиками, потому что так можно вызвать исключение, обработать его и сразу же внутри этого класса ошибки объяснить, что произошло, какие параметры были вызваны, что можно сделать, предотвратить или продолжать какую-то работу дальше, то есть вызвать там внутреннюю функцию, которая поможет нам, например, подскажет то, как мы можем обойти эту ошибку.

**Давайте рассмотрим пример(рис.6).**

Здесь мы сделали свой класс, который назвали ProZero. Он наследуется от класса Exception, но тут просто написано ключевое слово pass, то есть этот класс нечего особенного не делает, но так как он наследуется от Exception, то это уже исключение, а у него вшито внутрь методы и функции, которые есть у класса Exception. Далее здесь написала функция def, которая принимает на вход 2 параметра a и b. Если b у нас равно 0, то генерируется такое исключение, которое создаётся от этого класса ProZero (“Деление на нуль невозможно”)(рис.7), то есть сейчас мы ничего не изменили, то есть мы точно так же генерируем наши исключения, только мы используем не вшитый внутрь python исключение, а сделали своё. Иначе, если у нас b не равно 0, то мы возвращаем результат деления, то есть деление, где мы сделали свой собственный класс ошибки, если вдруг у нас b будет равно нулю, и ещё описание подробное на русском.

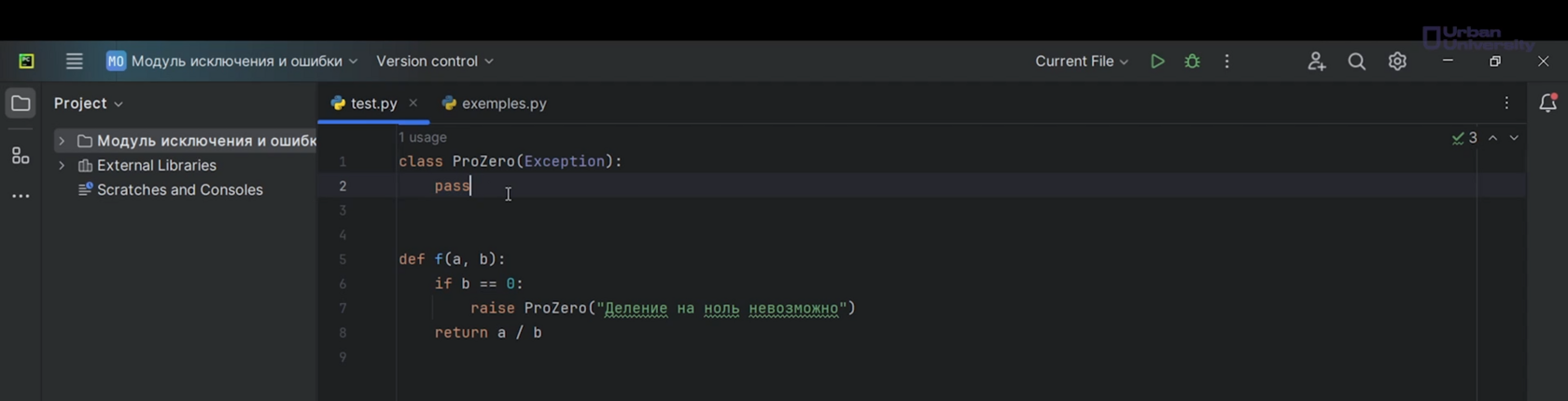


Рис.6

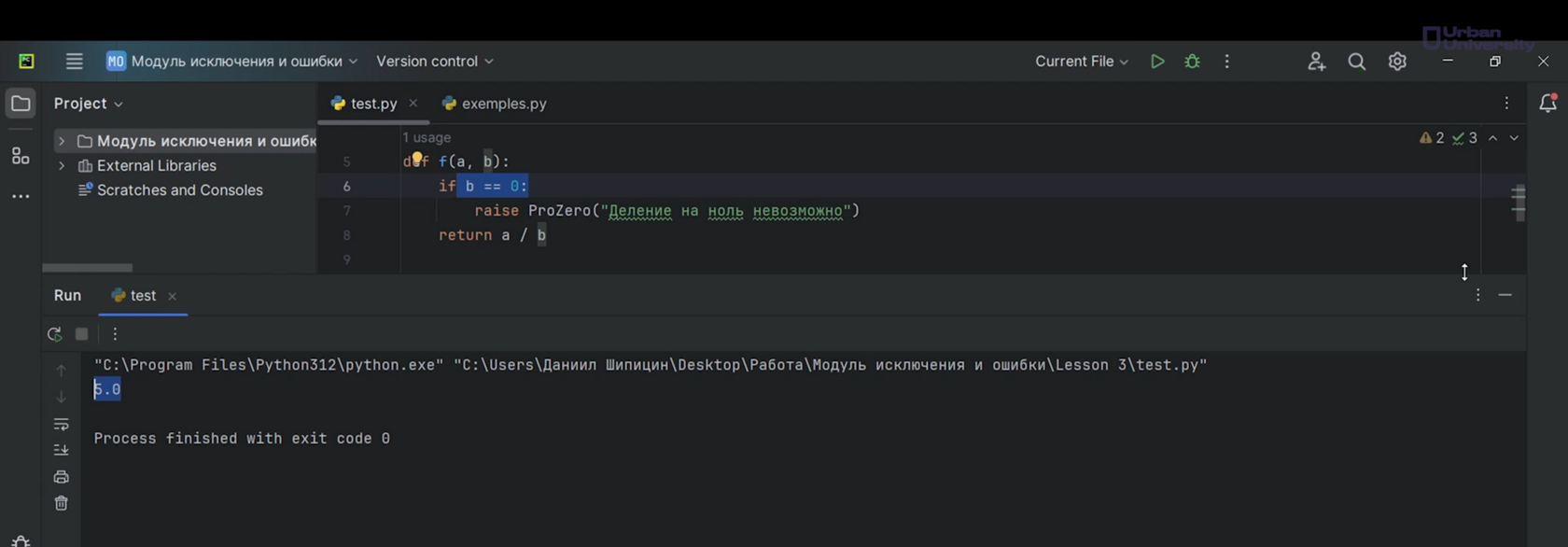


Рис.7

Давайте попробуем напечатать результат деления функции f от a: 5 и b: 1(рис.8). Сейчас у нас не сгенерирует исключение, потому что b не равно 0, программа просто успешно выполнится, но если b сделать 0, то тогда у нас появится Traceback, потому что мы не перехватили нигде исключение.

**Давайте посмотрим, что здесь у нас написано, читаем снизу наверх:**

* класс ошибки ProZero: Деление на нуль невозможно.
* далее на каком месте возникла эта ошибка, но в принципе логично, на каком, так как мы его сами генерируем, поэтому оно возникает на месте raise. Это произошло в функции f, когда мы пытались вызвать функцию f, то есть это наш стек вызова функции.

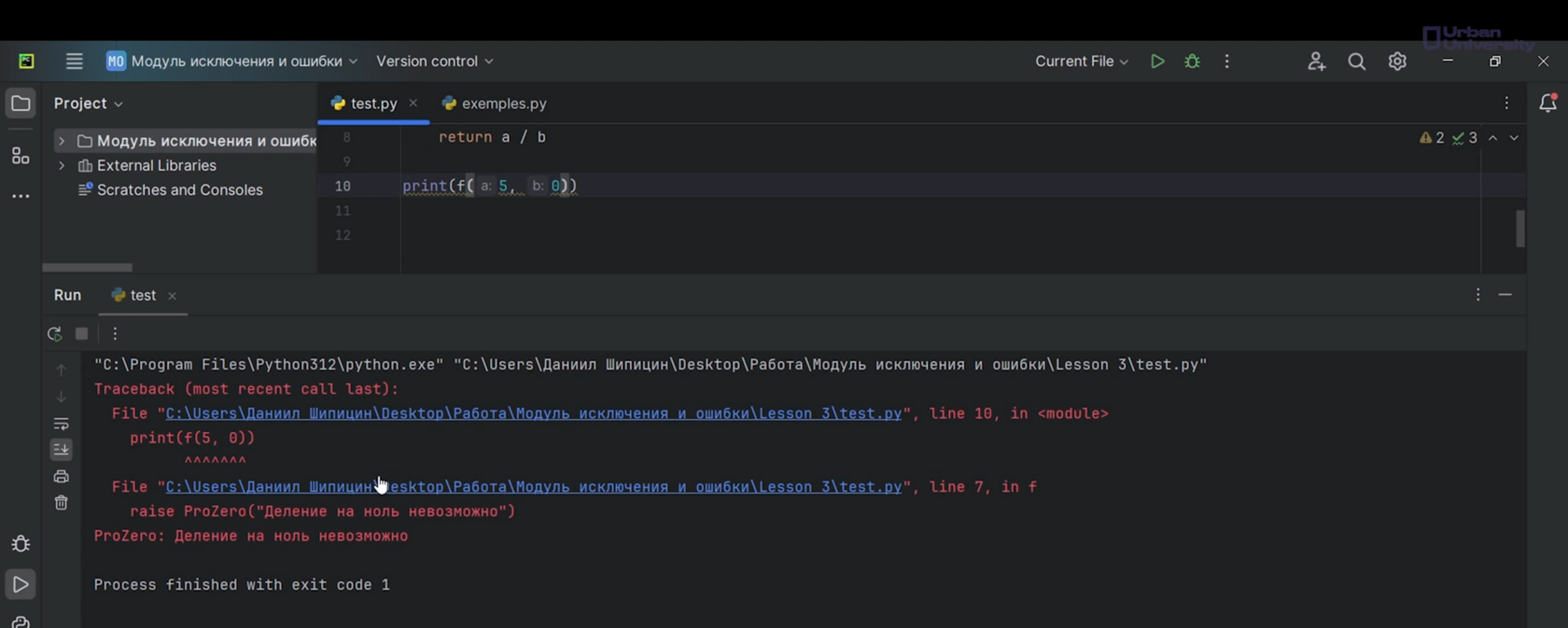


Рис.8

**Если мы посмотрим всё это в режиме отладки.**

Сейчас мы просто находимся на строчке номер 10, идем внутрь этой функции смотрим, как она работает: b=0, а=5(рис.9), значит у нас генерируются наше исключение ProZero и мы сразу же выходим из этой функции, потому что, как только сгенерировалось исключение, мы пытаемся отловить на нашем уровне, здесь не можем, поэтому выходим из нашей функции, но так как мы его и не ловим его на внешнем уровне, то у нас появляется окно трассировки(рис.10). Вот так мы можем делать свои собственные классы.

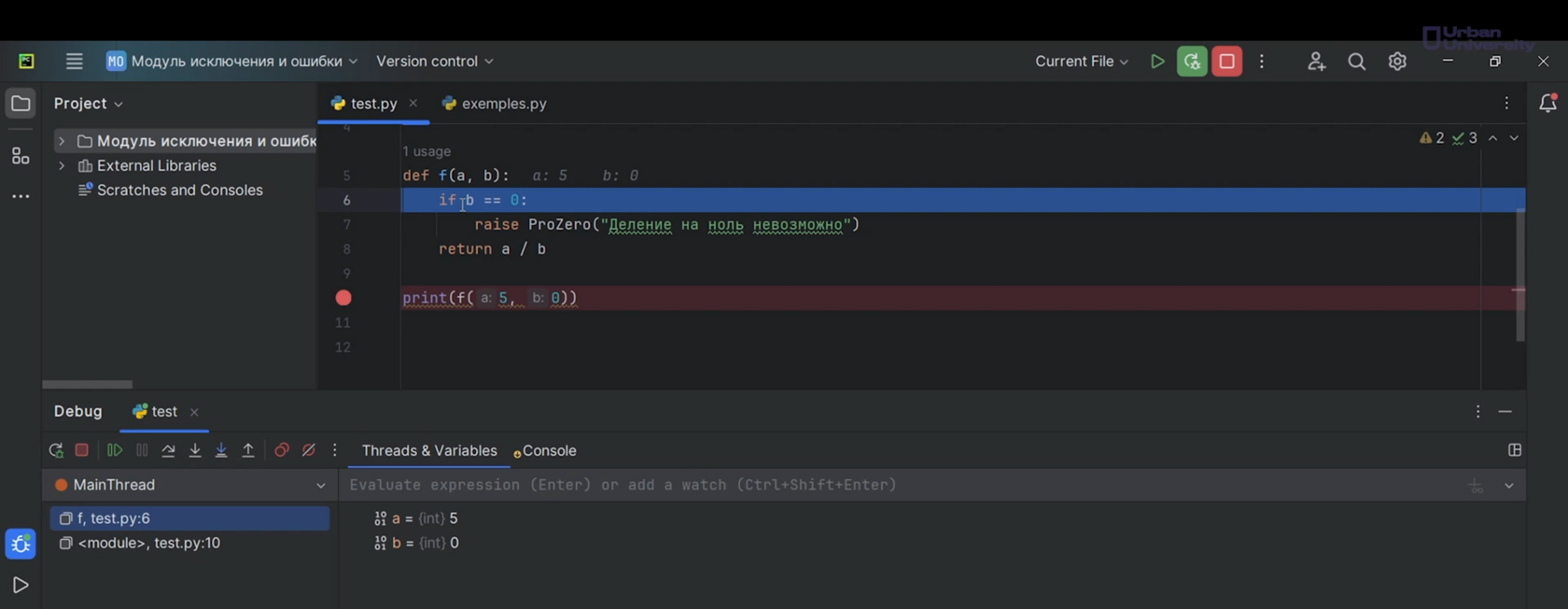


Рис.9

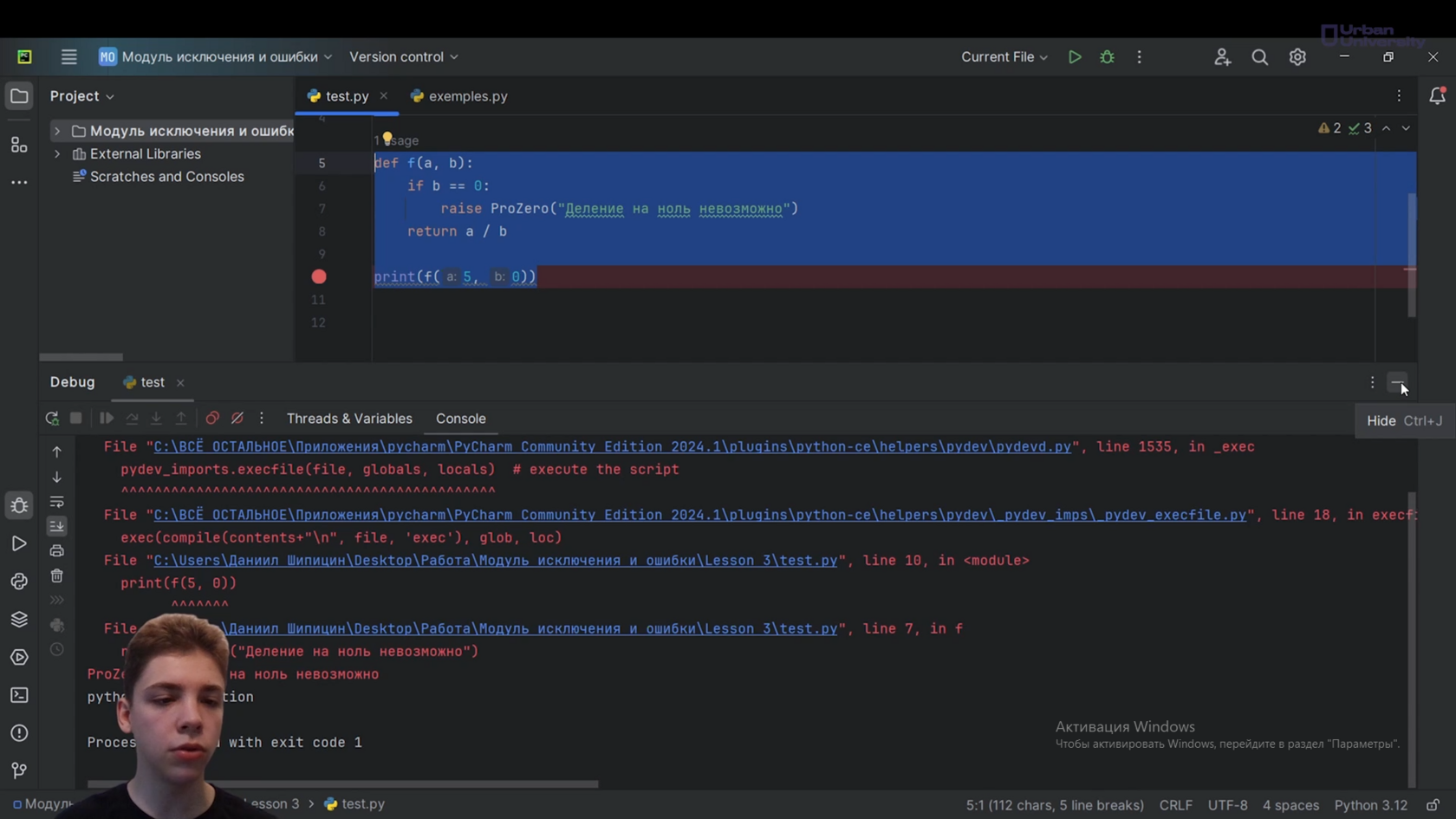


Рис.10

**Давайте посмотрим завершающий пример(рис.11)**, где немножечко улучшили наш класс ProZero, и теперь он работает более универсально, более интересно. Как работает эта программа?

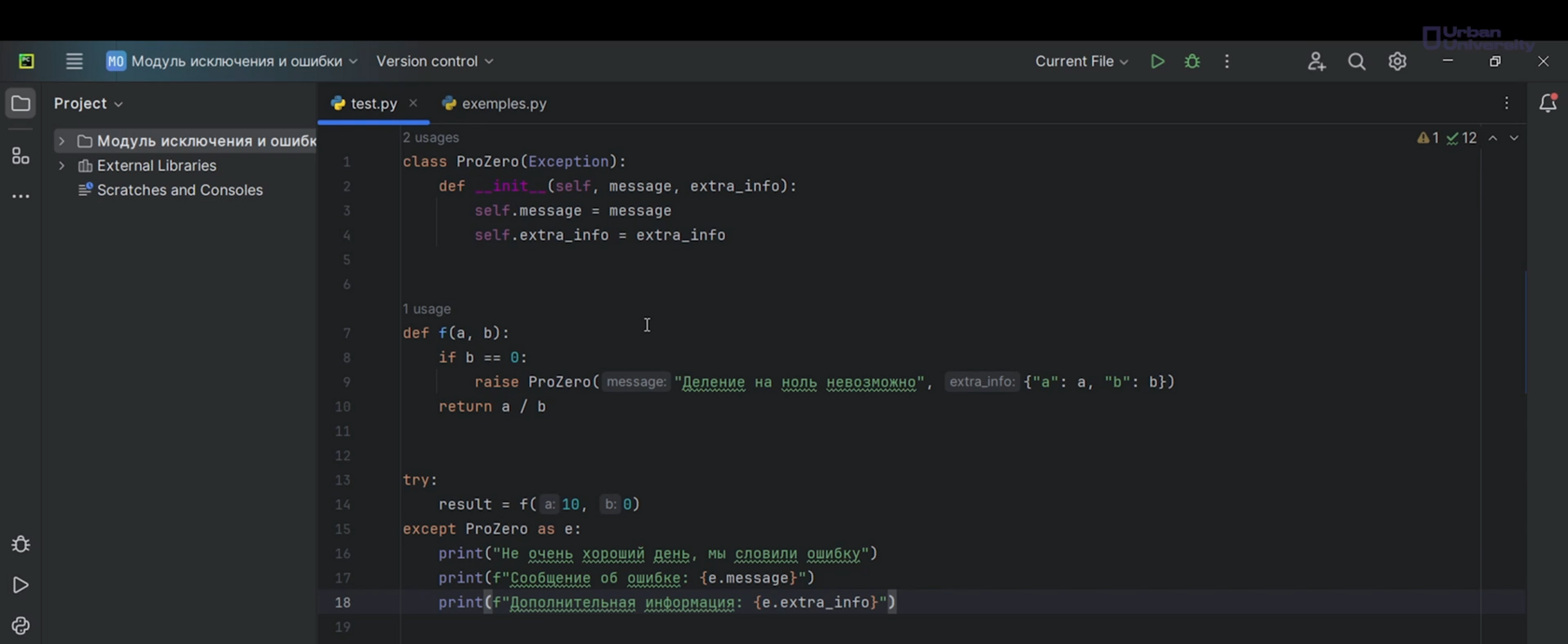


Рис.11

Если у нас b=0 и мы тогда перехватываем исключение ProZero, сохраняем её в переменную е и потом выводим:

* “Не очень хороший день, мы словили ошибку”
* “Сообщения об ошибке: {e.message}”, здесь будет написано сообщение, то есть пояснение к ошибке
* “Дополнительная информация: {e.extra\_info}, здесь будет передана дополнительная информация.

Как только у нас будет b=0, у нас выводится сообщение(рис.12):

“Не очень хороший день, мы словили ошибку  
Сообщение об ошибке: Делении на нуль невозможно  
Дополнительная информация: {‘a’: 10, ‘b’: 0}”.

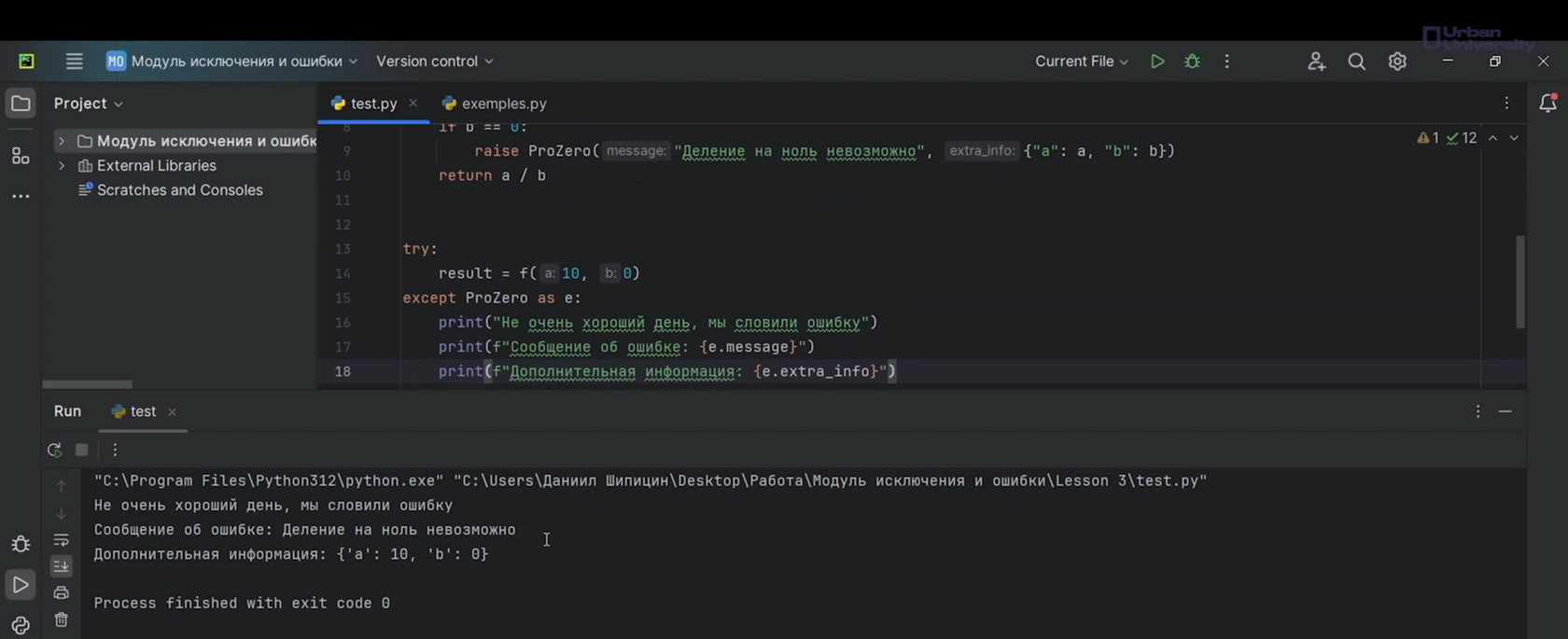


Рис.12

То есть мы смогли перехватить нашу ошибку и помимо того, то, что мы написали какие-то дополнительные сообщения, мы ещё отдельно поработали с классом ProZero, то есть не писали какие-то дополнительные функции, с помощью которых мы можем найти дополнительную информацию об этой исключительной ситуации, а в самом классе ProZero всё это прописали.

**Давайте посмотрим, как это было сделано.**

Точно так же у нас сделан класс ProZero, который наследуется от базового класса Exception, но просто не написано ключевое слово bass, а сделали метод init, в котором естественно, мы продаём аргументы: self, message, extra\_info. Message - это и будет наше подробное описание ошибки, то есть 1 аргумент - “Деление на нуль невозможно”, а 2 аргумент - extra\_info, в данном случае это словарик, в котором сохранены наши аргументы. При создании этого аргумента вызывается метод init и у нас создается объект message, точнее не объект, а атрибут объекта и создаётся у нас другой атрибут объекта extra\_info.

Когда у нас вызывается наша функция и вдруг случилось что b=0, тогда мы генерируем наше исключение ProZero, в котором мы передаём 2 параметра: первое - “Деление на нуль невозможно”, а именно message, и второе - extra\_info, мы можем всё что угодно написать, но мы здесь написали словарик. Если такого не произошло, то возвращаем результат деления a на b. Если бы у нас не было здесь конструкции try/except, то у нас бы просто вызвалось окно трассировки, но мы поместили вызов этой функции в блок try, а блок expect перехватил возможное исключение, а именно ProZero, сохранил его в переменную е(рис.13), вывел строчку “Не очень хороший день, мы словили ошибку”, с помощью атрибута e.message вывели сообщение “Деление на нуль невозможно”, с помощью атрибута e.extra\_info вывели дополнительную информацию ‘a’: 10, ‘b’: 0.

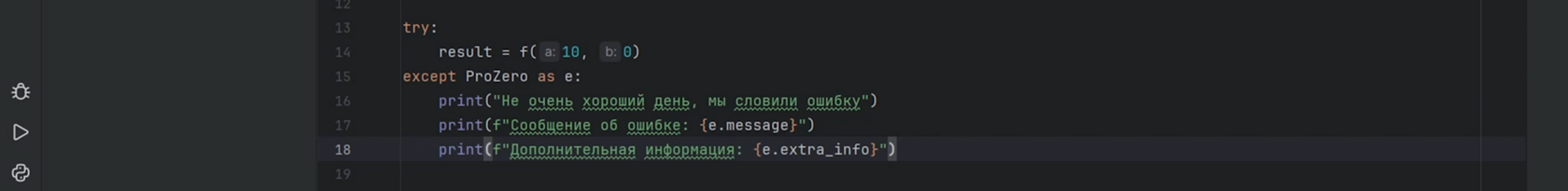


Рис.13

Если бы мы передали b=1, то у нас просто бы вывелся результат деления на один(рис.14). Как только здесь сгенерировалось исключение, оно перешло на уровень блока try, в функции result сгенерировалось исключение, вышла из выполнения этой функции и перекатилась уже на уровне немножечко повыше.

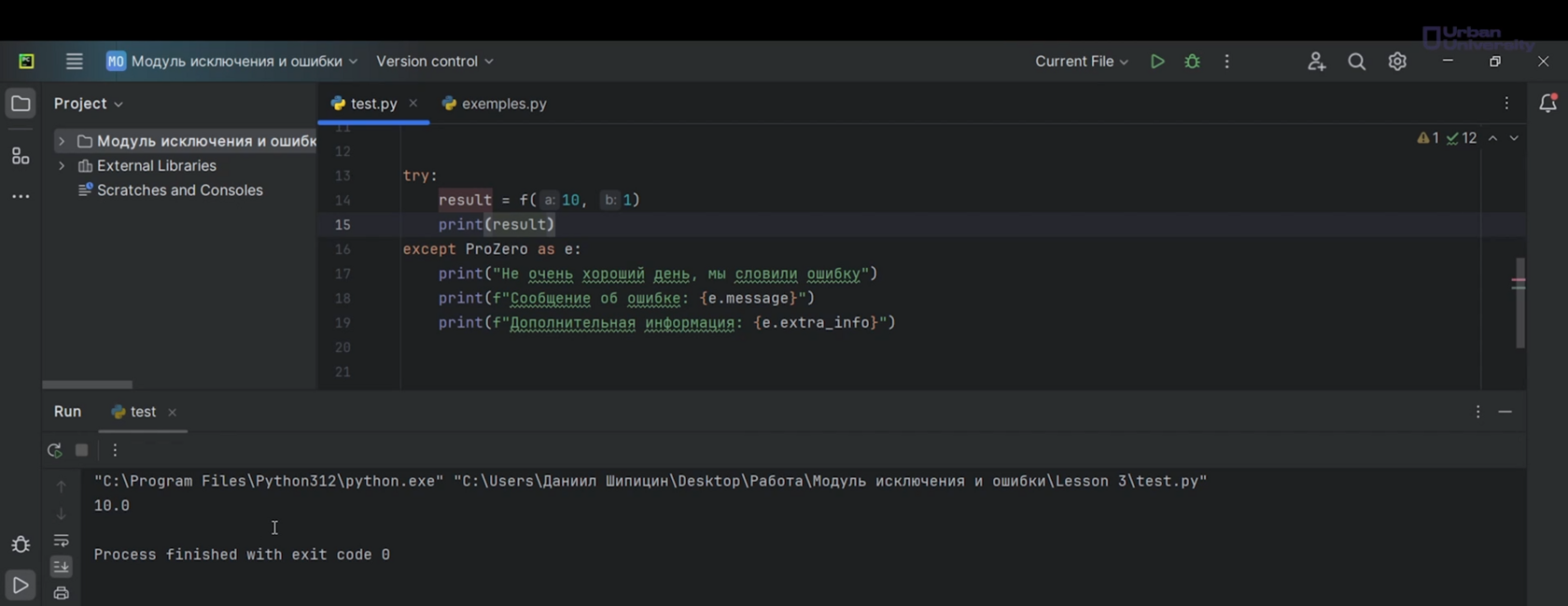


Рис.14

Таким образом вы можете создавать собственные классы ошибок, их естественно можно еще больше улучшать, но если у вас не самая сложная программа или вы пишете программу на каком-то framework, то скорее всего, там есть собственные исключительные ситуации или классы исключений. Нужно читать документацию и понять то, как вы можете ей пользоваться.