**2023/10/08 00:00|Лекция. Произвольное число параметров**

## Произвольное число параметров

На прошлом занятии мы рассматривали функции с параметрами по умолчанию. Если вспомнить, то функции с параметрами по умолчанию могут вызываться с переменным количеством аргументов. Мы можем передавать в них, например 1, 2, 3, но в зависимости от того, сколько было объявлено параметров в момент объявления функции. Нередко возникает ситуация, когда нам необходимо передавать произвольное число параметров и главная проблема заключается в том, что мы не можем заранее предугадать, сколько параметров будет передано. **В таких случаях, если нам необходимо какое-то произвольное число параметров, для позиционных мы можем использовать знак “звездочки”.**

Создадим просто функцию, назовем ее test\_func и напишем \*params. Эта функция просто будет выводить эти параметры. Давайте ее вызовем и посмотрим, как это работает, если мы в нее ничего не передадим. Запускаем. Увидели пустые скобки(рис.1). Если захотим передать в нее несколько параметров, при запуске получим все параметры, они все отобразились в круглых скобках(рис.2).

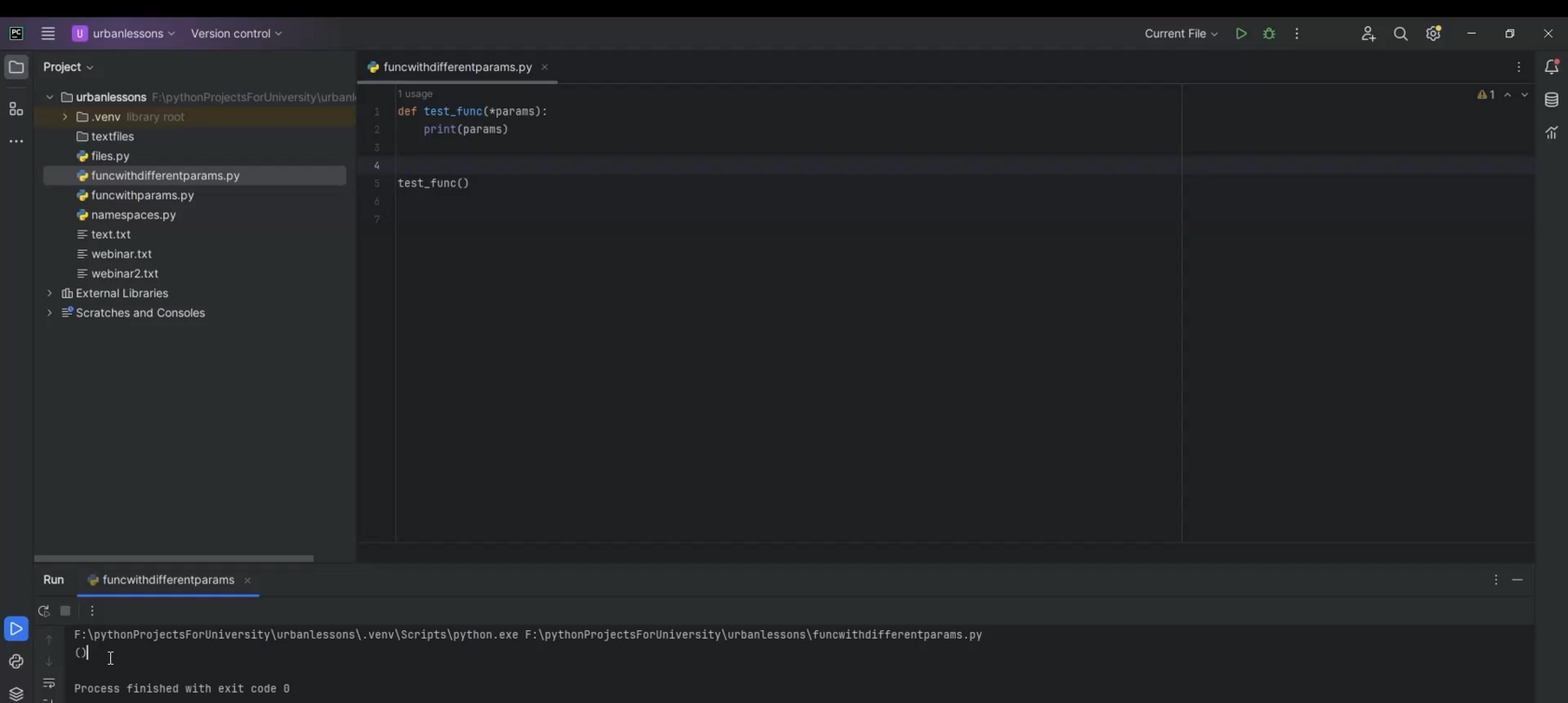


Рис.1

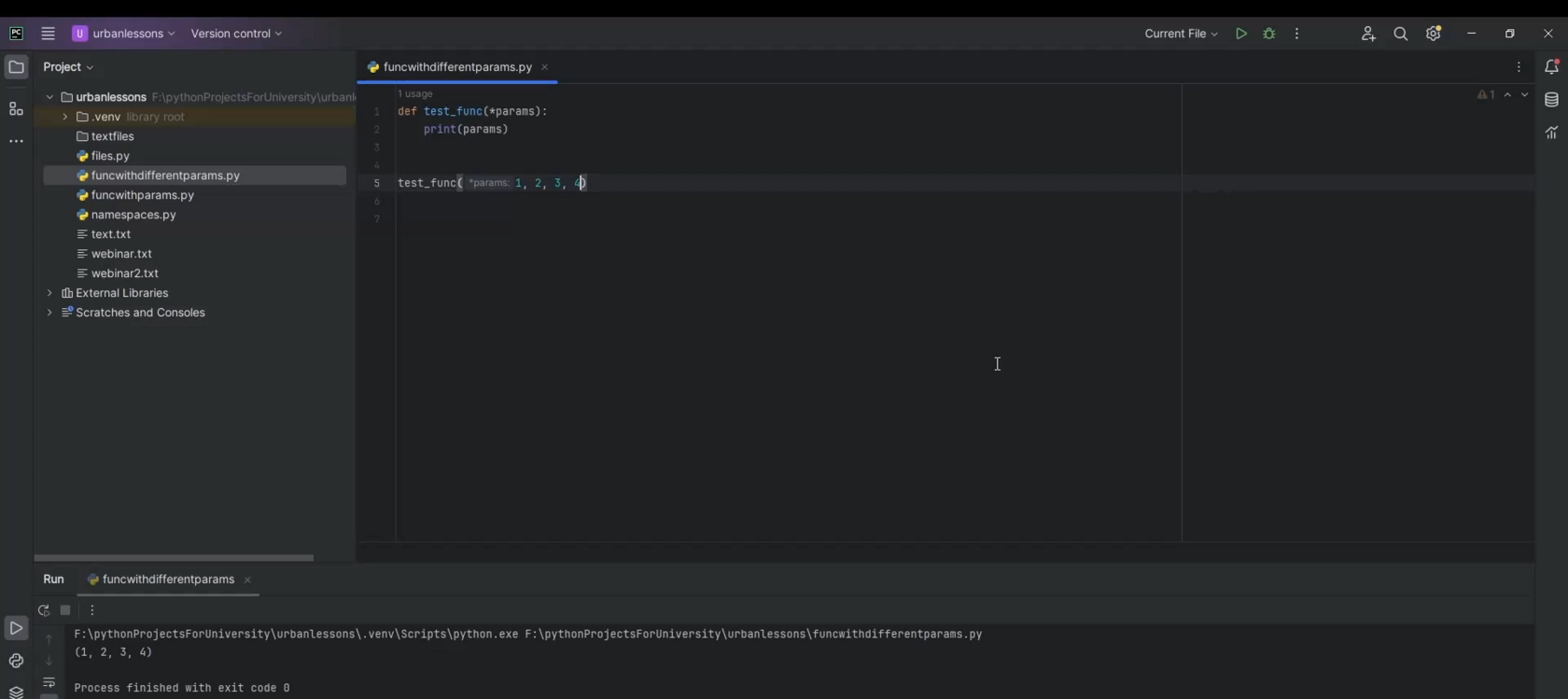


Рис.2

Когда мы используем “звездочку” и пишем название нашего параметра, технически это выглядит как кортеж, хотя мы просто записываем в момент вызова функции какие-либо параметры через запятую. Если мы хотим убедиться и по подробнее рассмотреть, давайте немного эту функцию изменим. Пускай останется test\_func(\*params), но далее выведем текст print(“Тип:”, params), также воспользуемся функцией тайп print(“Тип:”, type(params)) и выведем уже соответственно, аргумент print(“Аргумент:”, params)(рис.3). При запуске получается тип, значение которого мы будем передавать в эту функцию - это кортеж, и все эти params, которые мы передаем, все параметры будут являться кортежем.

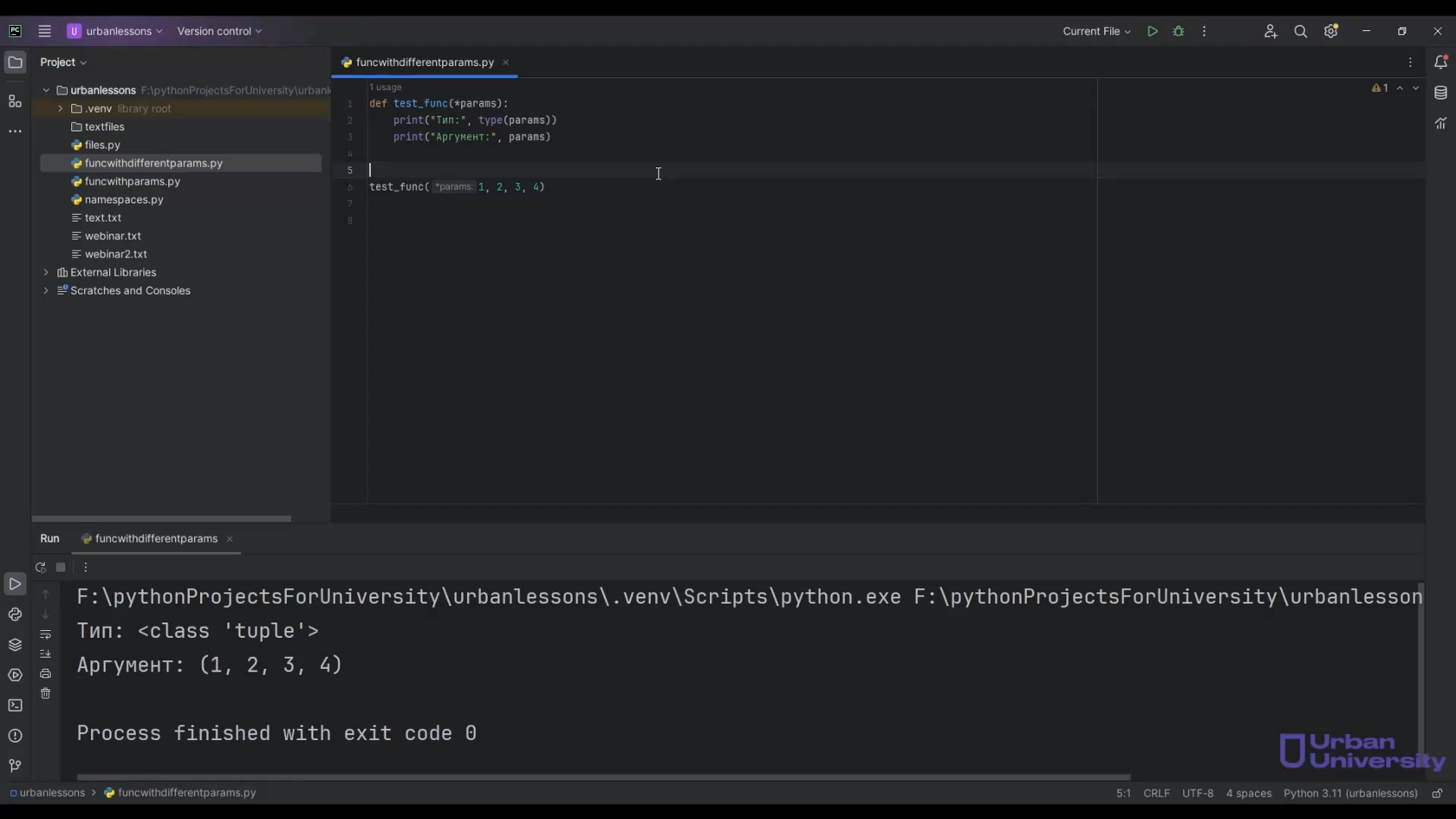


Рис.3

Мы можем использовать по-разному такие функции, например, если нам необходимо посчитать сумму чисел, при этом мы не знаем, сколько пользователь передаст в эту функцию чисел. В таких случаях мы будем использовать такую же запись.

Давайте создадим функцию, назовем ее summator. Он будет принимать какие-то значения, параметр \*values. Будет какая-то переменная s, которая будет хранить изначально число 0(рис.4). Впоследствии мы будем пробегаться по вот этим значениям, то есть по кортежу, которые мы будем передавать и прибавлять значение из этого кортежа к этой переменной, и в конечном счёте эта функция будет возвращать нам эту s. Создадим цикл for который будет перебирать эти значения. Также мы можем к этой переменной s прибавлять переменную i, которая находится внутри цикла for. После того как цикл отработал, мы будем возвращать переменную s(рис.4).

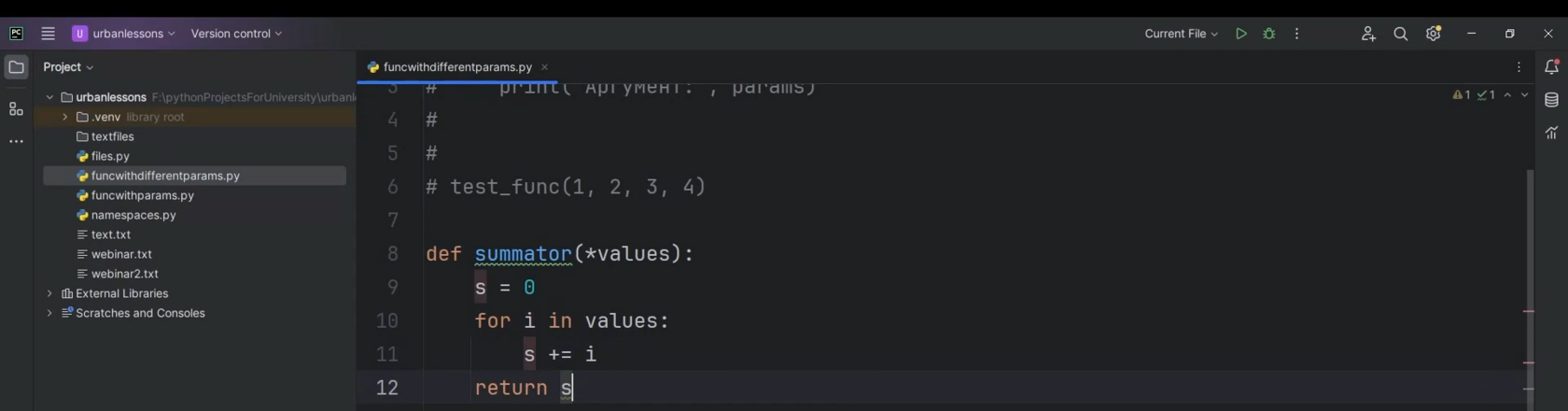


Рис.4

Давайте выведем результат работы функции summator. Напишем 1, 2, 3, 4, запустим, получили 10: 4+3=7; 7+2=9; 9+1=10(рис.5). Все верно, функция посчитала правильно, то есть, по сути, когда мы ставим “звёздочку”, мы говорим, что мы будем передавать произвольное число каких-то позиционных параметров.

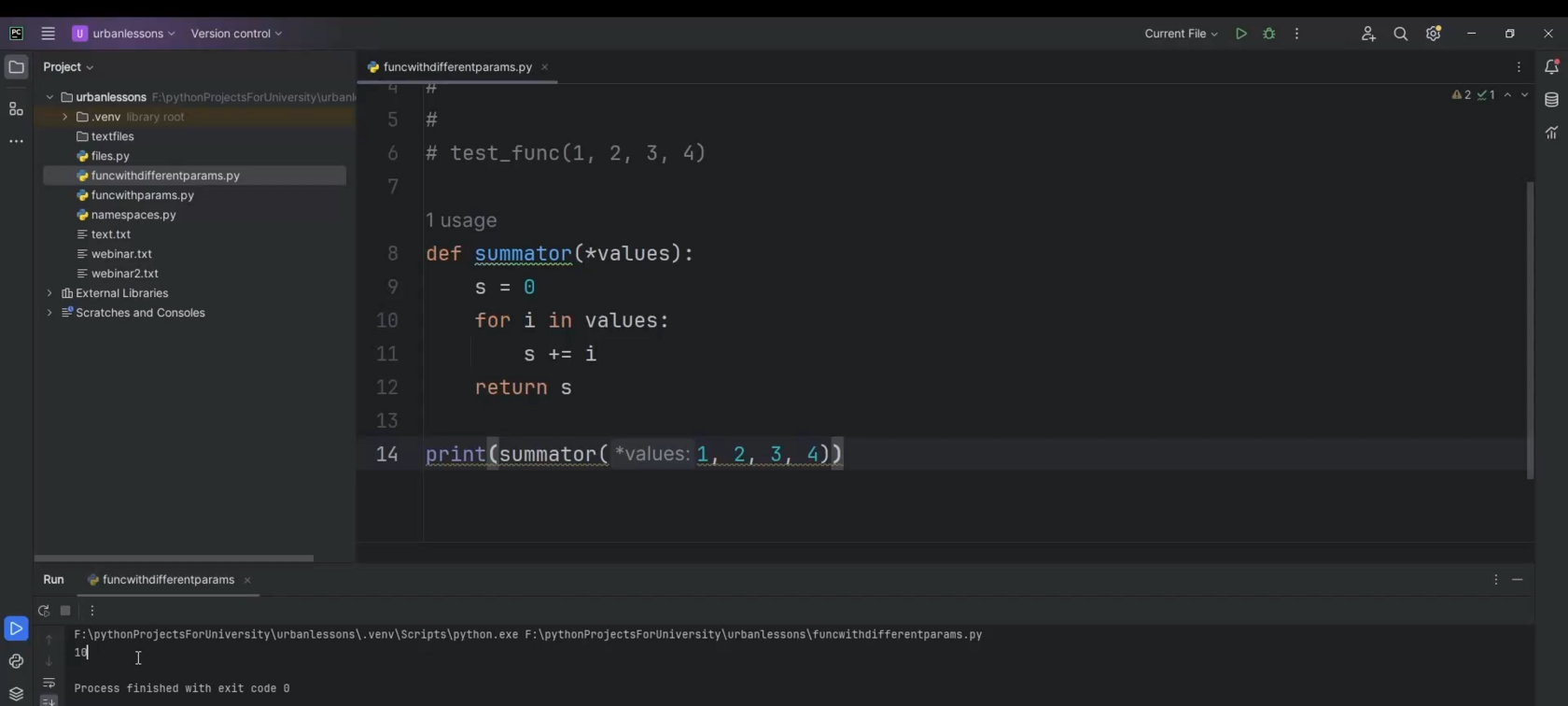


Рис.5

Такие параметры можно комбинировать, то есть мы можем использовать здесь обычный позиционный параметр, например текст(txt). Создадим f строку, она будет принимать параметр txt и переменную s с суммой.  
Буква «f» в Python — один из ключевых элементов синтаксиса, который используется для форматирования строк. Благодаря этой букве можно создавать строки, в которые можно вставлять значения переменных.  
Для использования буквы «f» в строке необходимо перед строкой указать префикс «f». После этого в фигурных скобках можно указывать название переменной или выражение, которое будет вставлено в строку.  
Получается функция вернёт строку, в которой будет значение параметра txt и сумму. Тогда мы в момент вызова первое значение, которое передадим, оно будет передано в параметр txt. Сюда мы передадим строку “Сумма чисел:”. При запуске мы получим “Сумма чисел: 9”(рис.6). У нас есть возможность комбинировать.

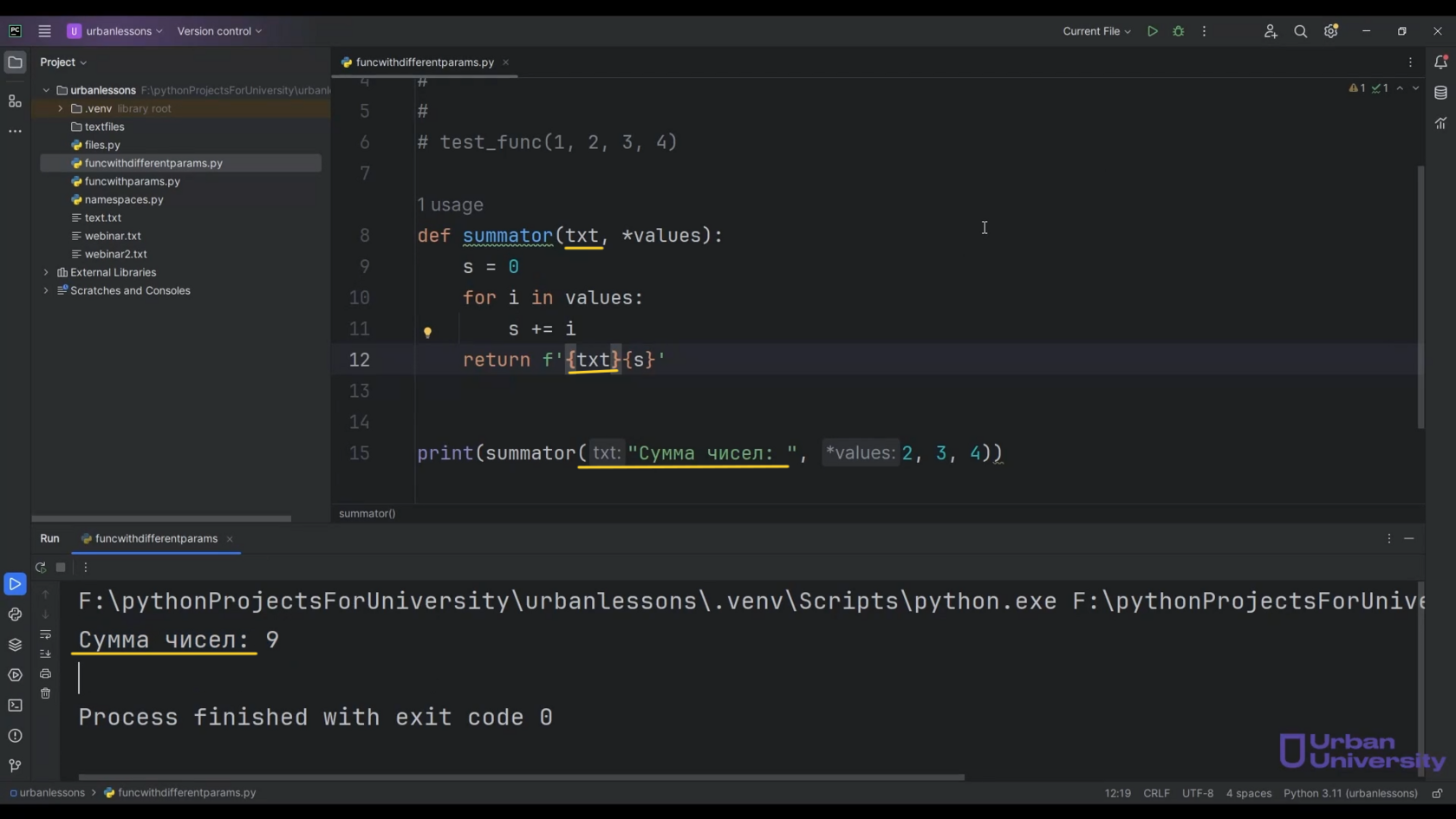


Рис.6

Также мы можем сюда передавать еще и значения по умолчанию, например type=”sum”. Если сюда добавить вывод параметры type, который задан по умолчанию, то функция вернет сообщение такого вида: “Сумма чисел: 9 sum”(рис.7). Это результат перебора элементов из параметров, которые мы передали и параметр со значением по умолчанию. Если мы заходим его изменить, то можем написать type=”summator”. Запускаем, получаем “Сумма чисел: 9 summator”(рис.8).

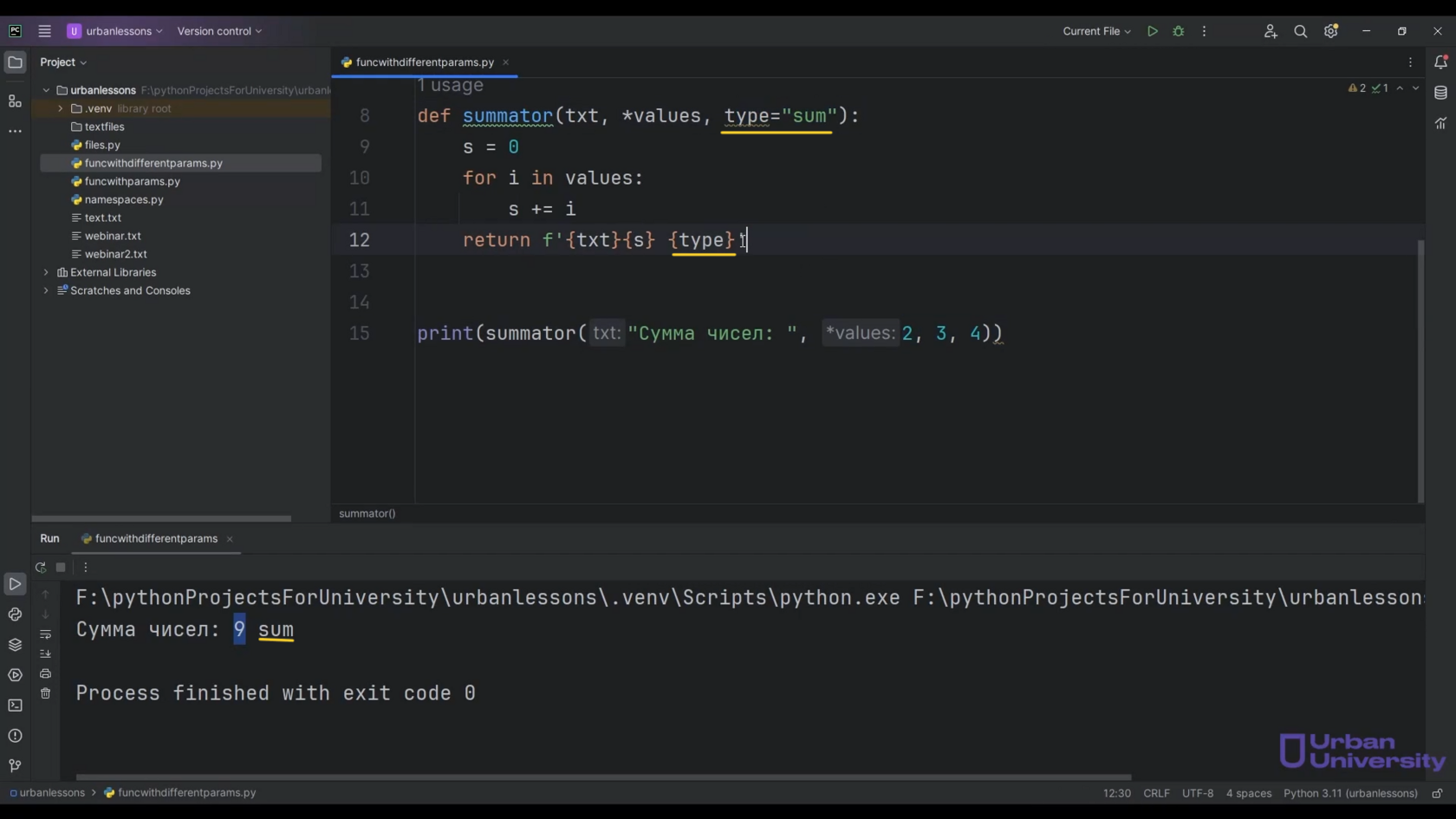


Рис.7

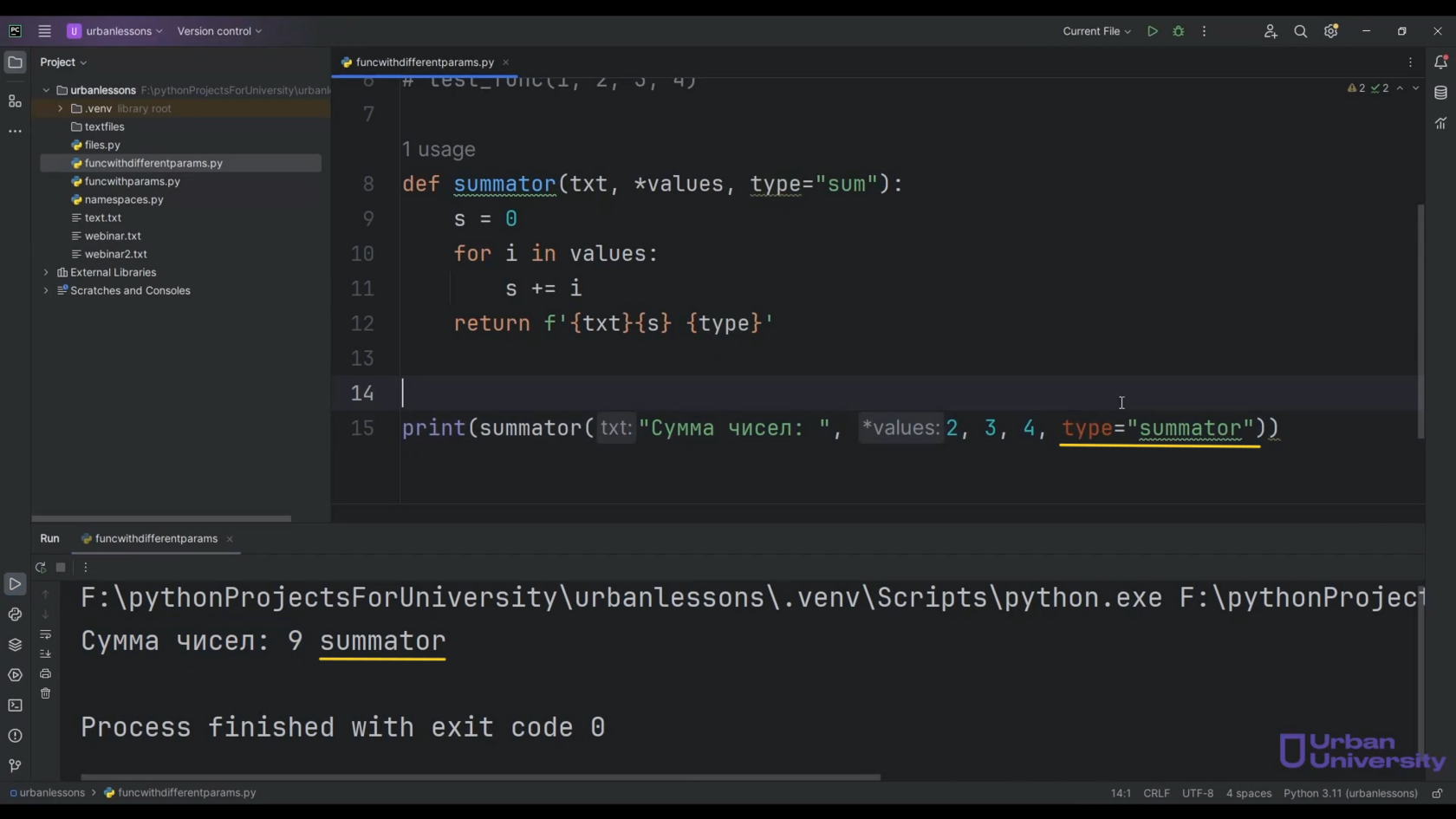


Рис.8

**Есть возможность использовать все виды параметров при объявлении функции как: позиционное, параметры с переменным количеством, именованные параметры - параметры со значением по умолчанию. В том числе есть возможность использовать и произвольное число именованных параметров.**

Например, создадим функцию, которая будет собирать информацию о человеке. Назовём её info и здесь, с двумя звездочками укажем имя параметра values и посмотрим, как это будет реализовано. Скопируем принты с типом и аргументом, но вместо params будет values. Вызовем функцию info. Здесь наша задача передать уже именованные параметры, например name=”Denis”, course=”Python” и запустим(рис.9).

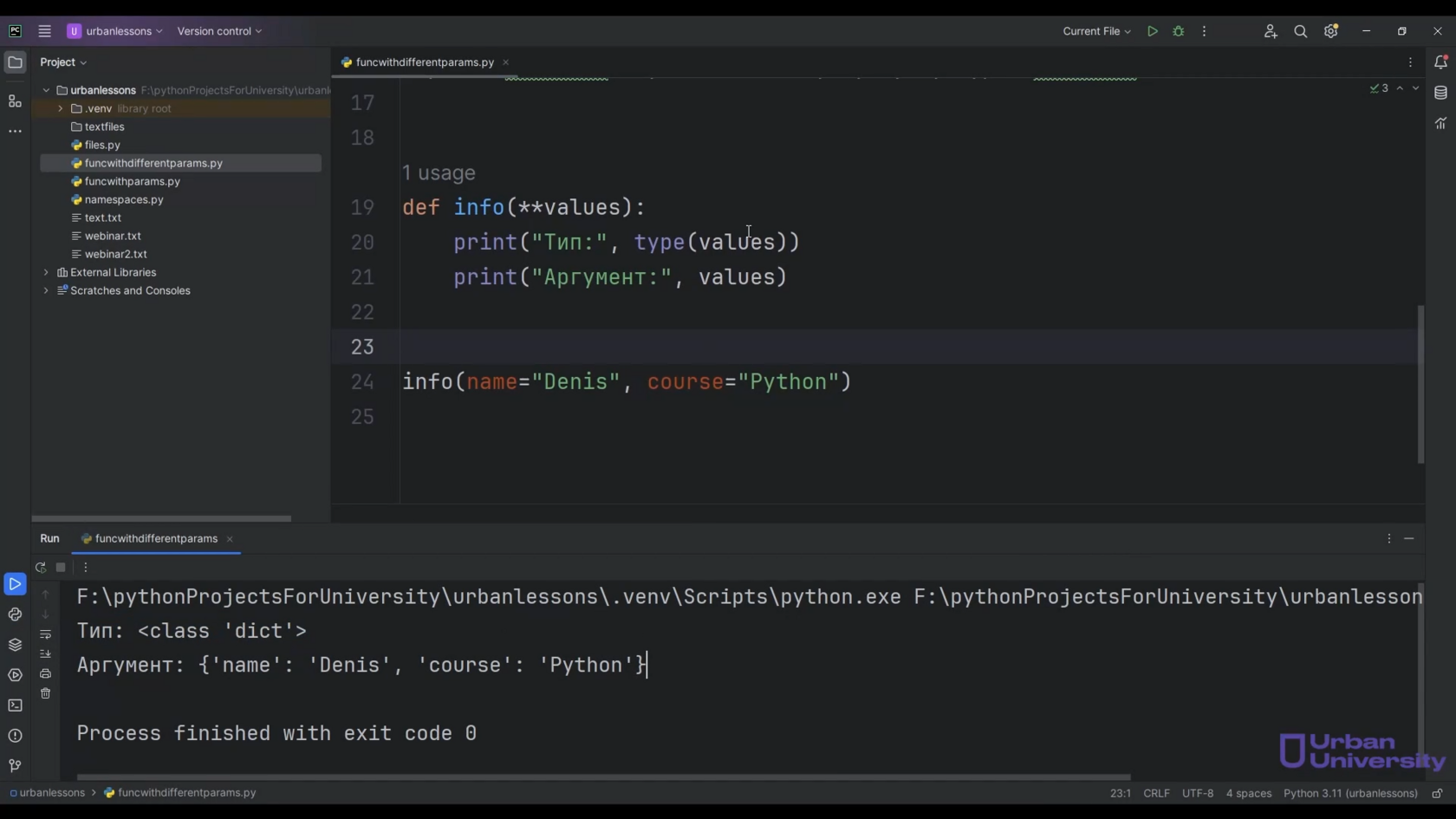


Рис.9

Здесь получается при использовании 2 звездочек, мы говорим о том, что не знаем, сколько будет передано именованных параметров. То есть это произвольное количество именно именованных параметров. Технически это будет реализовано в виде словарика.**Словарик**, как мы помним - это коллекция, которая хранит элементы парами: ключ и соответственно значение(рис.10). Мы указываем сначала ключ, ставим знак равно, и потом пишем значение, которое соответствует данному ключу. Получается это реализовано как словарь и передаётся как словарик, соответственно, мы можем использовать различные методы. Например, с помощью цикла for пройтись по этим параметрам и сделать вывод, который удобно читать: name=”Denis”, course=”Python”(рис.10).

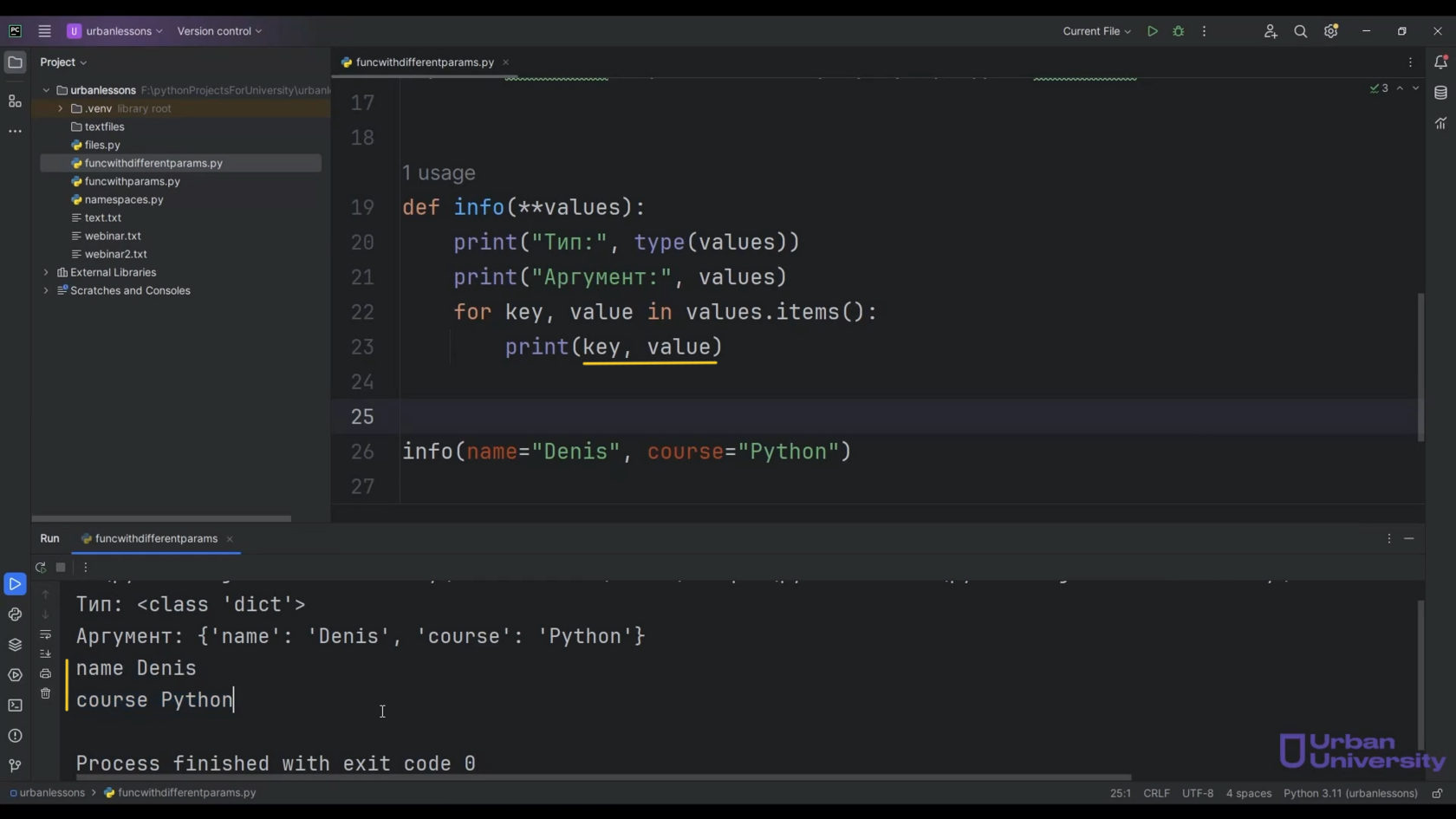


Рис.10

В том числе, если есть возможность использовать переменное количество именованных параметров, мы можем это комбинировать с переменным количеством позиционных параметров.  
**!**Единственное, что **позиционные параметры никогда не идут после именованных**, то есть даже если мы захотим написать одну звездочку и сказать, что мы будем передавать произвольное количество позиционных параметров, это не сработает(рис.11). Это делать необходимо перед именованными.

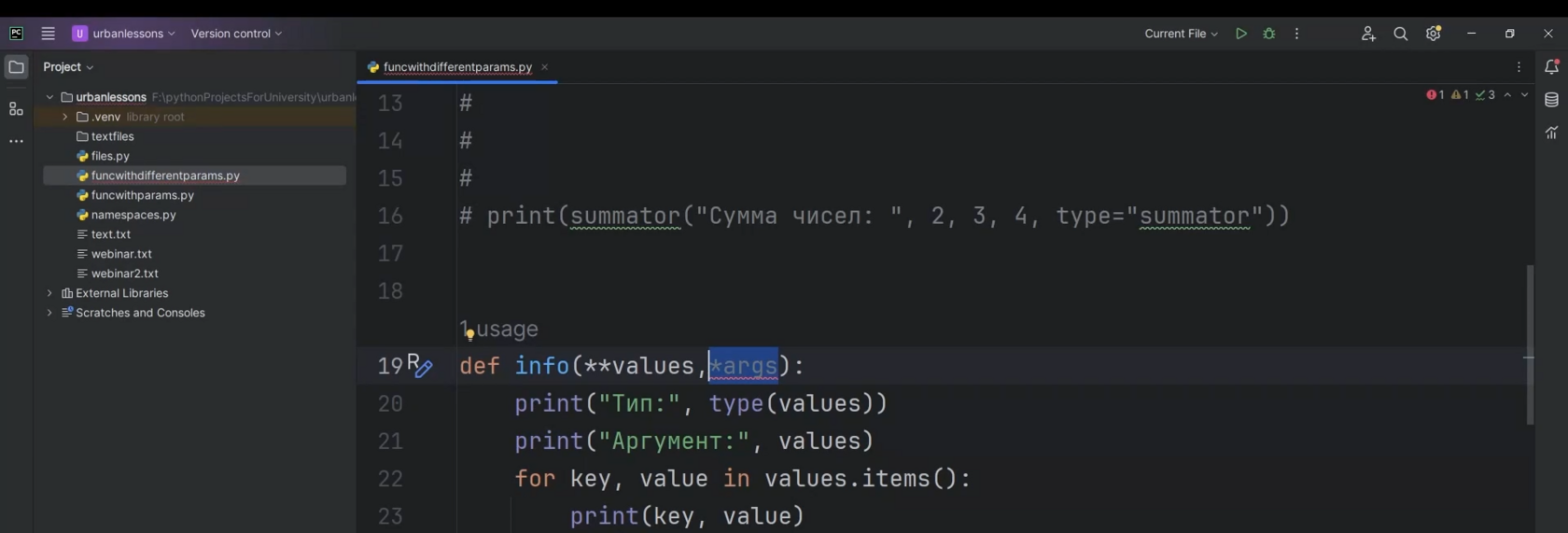


Рис.11

До параметра \*\*value будет также параметр, назовем его \*types. В момент вызова мы можем точно так же передавать, например, types(рис.12). Если мы передаем не именованные параметры, а позиционные, соответственно, они уже будут восприниматься немного по-другому. Если мы обратим внимание на последнюю строчку нашего вывода, здесь снова получаем кортеж из элементов(рис.13). Таким же образом, мы все это дело можем комбинировать со значениями по умолчанию.

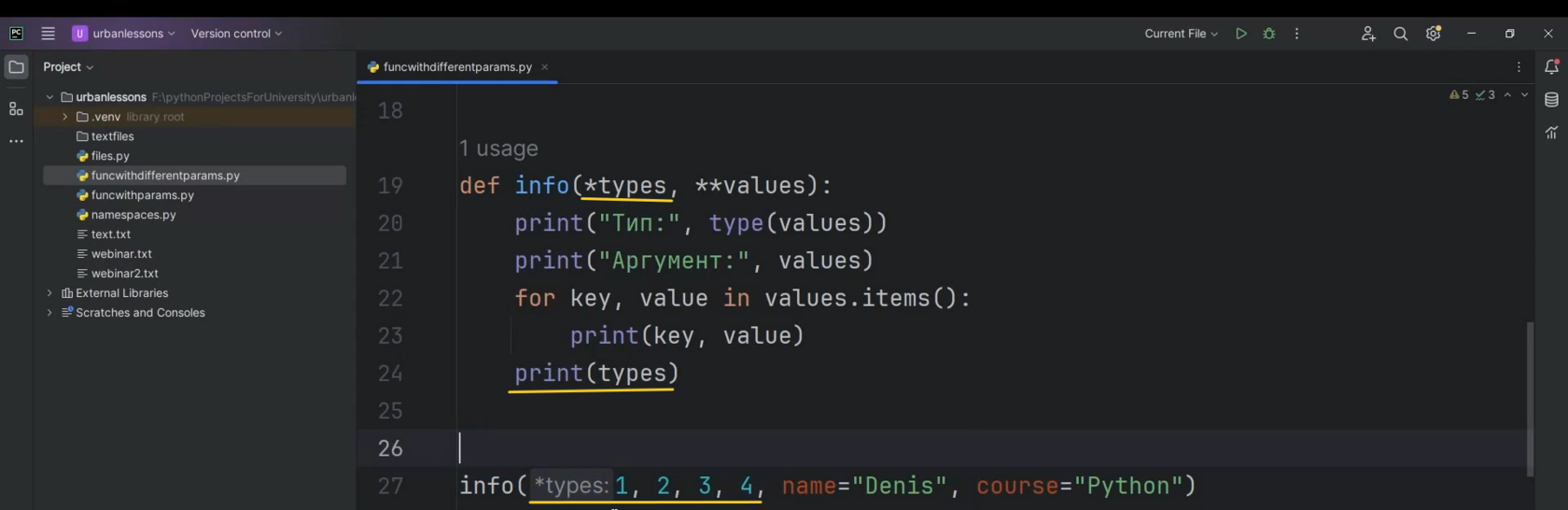


Рис.12

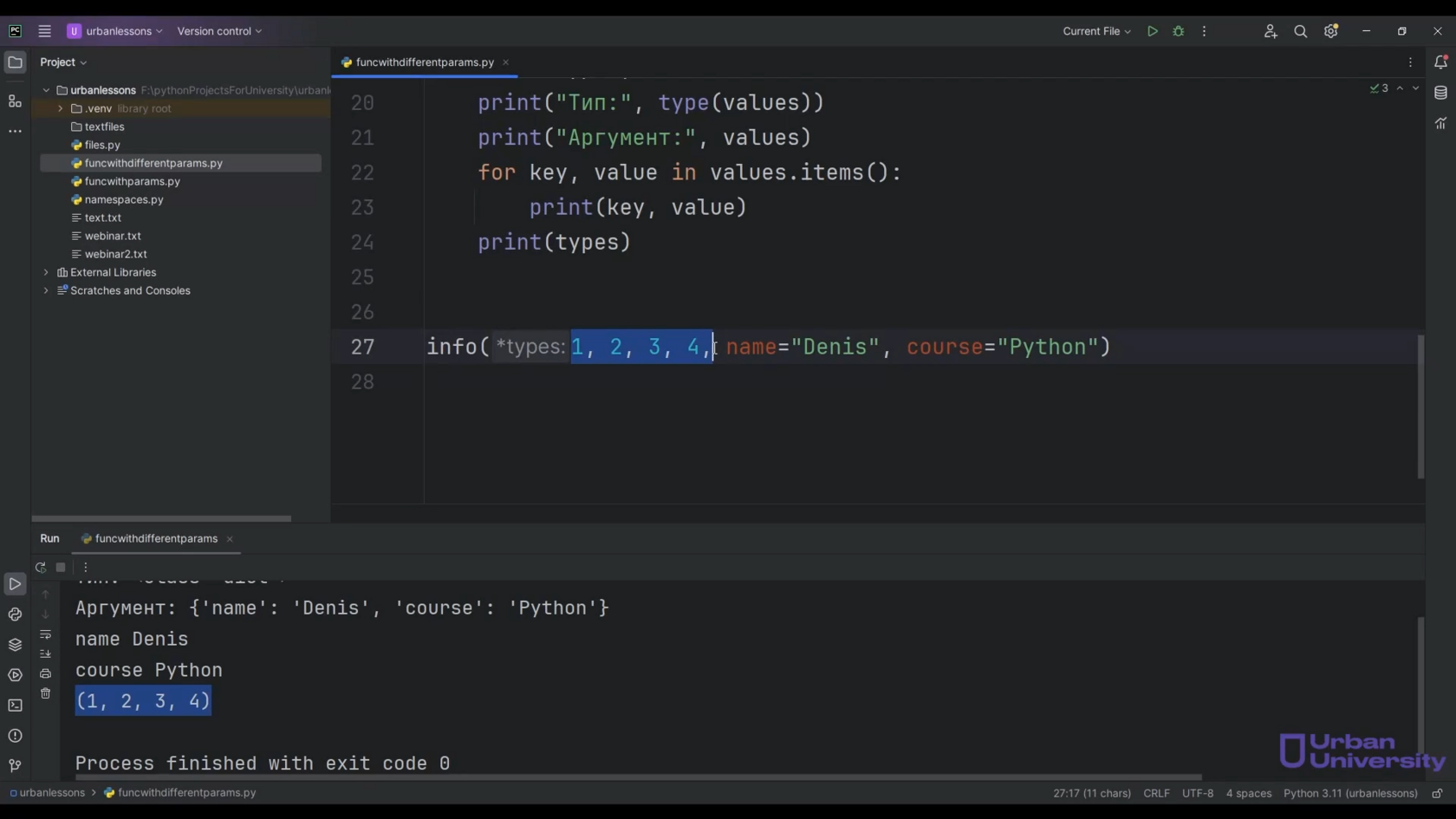


Рис.13

Только **обратите внимание**, что, если здесь идет переменное количество именованных параметров, то данное значение мы будем вставлять непосредственно перед произвольным количеством. Когда мы перечислили все конкретные параметры, только потом указываем, что будем использовать произвольное количество, то есть можем объединить все виды параметров в одном: обычный позиционный параметр, произвольное количество позиционных параметров, именованные параметры и произвольное количество именованных параметров. Если данный порядок будет нарушен, то не будет возможности просто объявить функцию.

В разных ситуациях это может пригодиться, например, когда при вызове функции один параметр зачастую принимает одно и то же значение, либо, когда нам необходимо передать какой-то позиционный параметр, который будет объявлять, допустим, тип действия у нашей функции. Если мы напишем на 1 месте какое-нибудь значение, например “пример использования параметров всех типов”, это будет восприниматься как обычный позиционный параметр. Именно value, который находился первым, мы его передали. Все последующие параметры позиционные будут относиться к \*types, то есть будут переданы в виде кортежа. Через name передали позиционный параметр. Name=”Den”(рис.14) только изменил его значение и следующие именованные параметры, которые мы будем передавать после него.

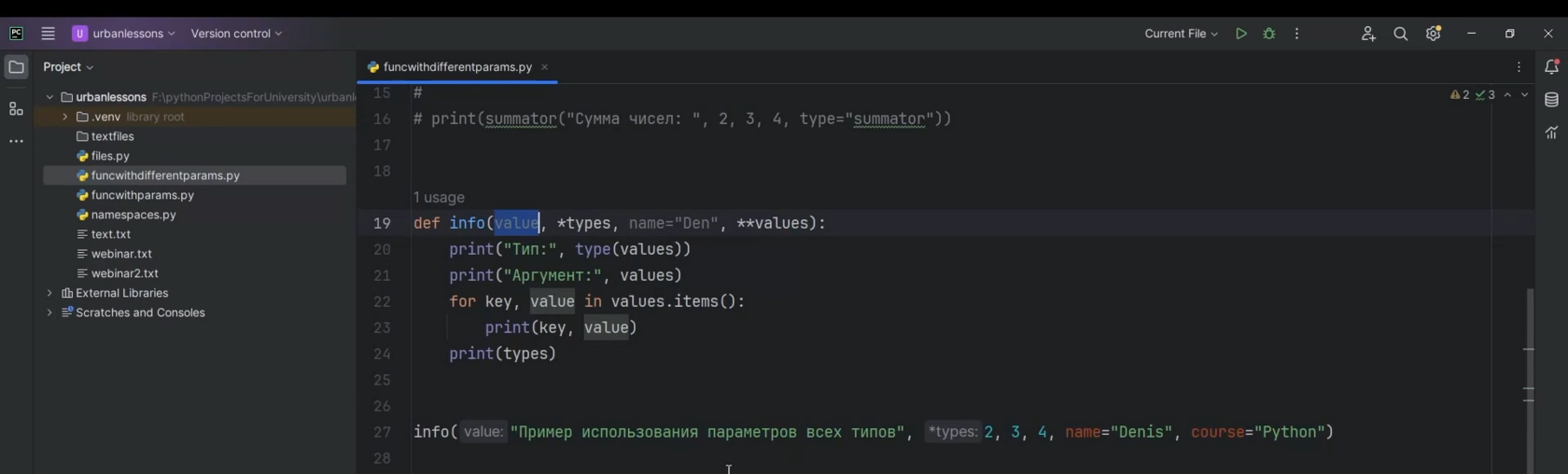


Рис.14

Видим также что name у нас совпадает с другим name. Перепишем их на name\_author=”den” и name\_author=”Denis”. Последующие именованные параметры name и course, которые мы будем передавать, будут относиться к \*\*values и передаваться в виде словаря. При запуске у нас выйдет(рис.15):

Тип:  
Аргумент: {‘name’: ’Den’, ‘course’: ‘Python’}name Den course Python (2, 3, 4)

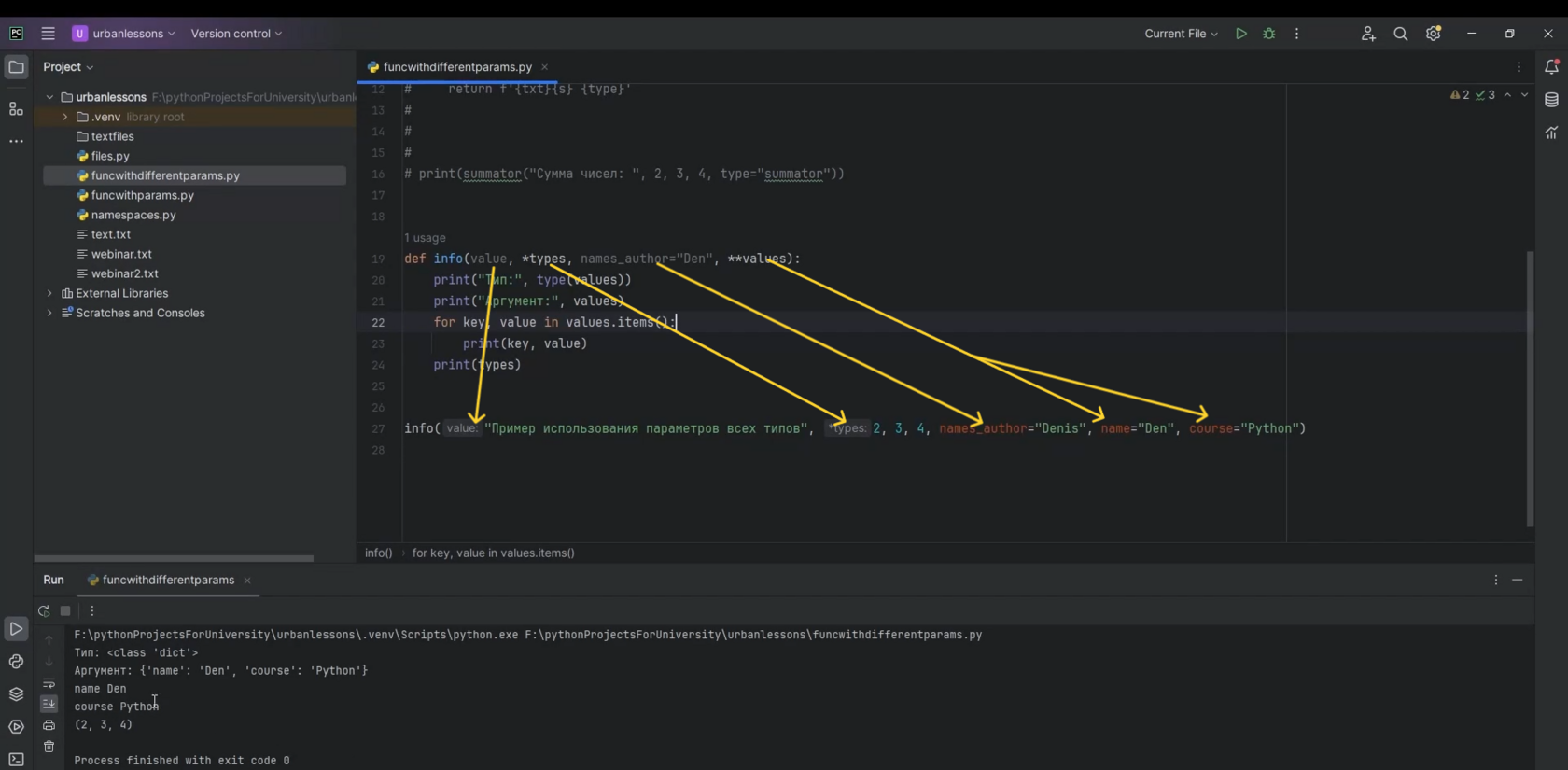


Рис.15

То есть print(“Тип:”) и print(“Аргумент:”) касаются значений, которые передали в последнюю очередь и цикл for перебирает их. Здесь важно просто сохранять порядок. Мы никогда не знаем, какого вида функцию нам необходимо сделать, и какая функция нам может пригодиться. Например, можно сделать что-то общее.

**Давайте как финальный заключительный пример** сделаем функцию, которая будет находить сумму чисел, но сделаем её немного необычной: она будет находить сумму чисел и сможет находить, допустим, сумму квадратов, сумму кубов, то есть принимать какую-то степень этих чисел. Назовем ее my\_sum, она нас будет принимать параметры - степень, какое-то произвольное количество позиционных параметров и именованный параметр txt.

Изначально она рассчитана на сумму чисел. В этой функции точно таким же образом, как и в примере ранее, создадим какую-то переменную, которая будет изначально хранить число ноль, чтобы мы потом все параметры могли к ней приплюсовать. Также создадим цикл for, только здесь мы будем использовать range() по длине вот этих параметров(рис.16).

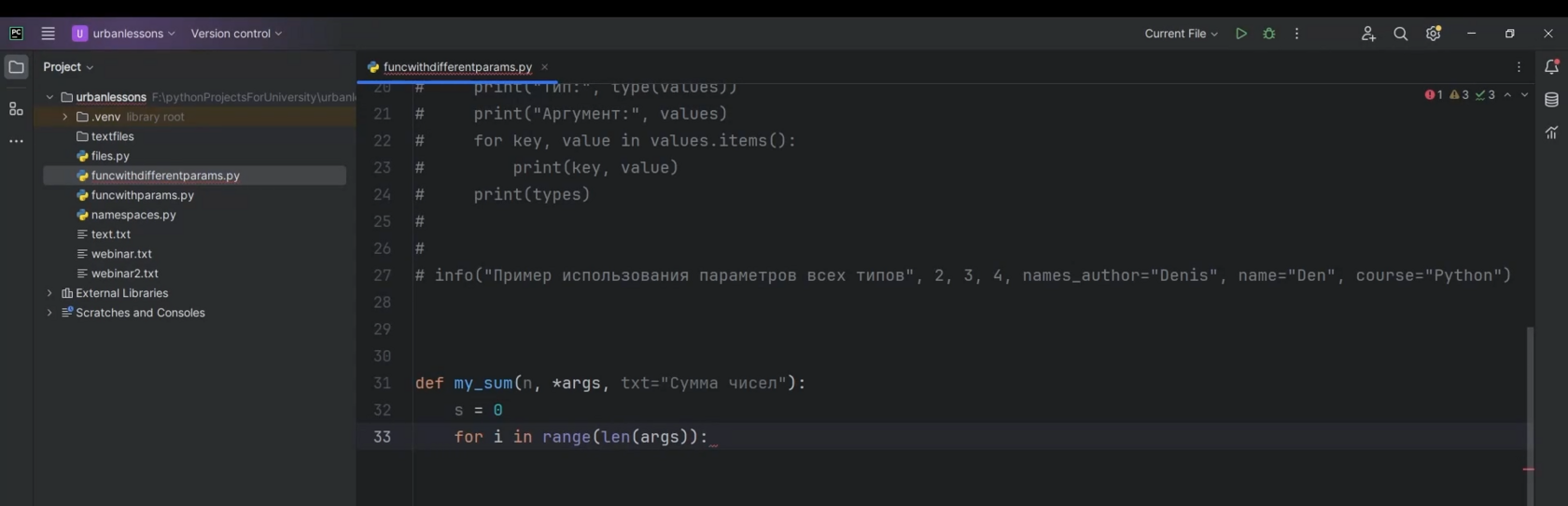


Рис.16

Если вспомните, они передаются в виде кортежа. Так мы будем обращаться к этим элементам по их индексам. Здесь мы будем брать из наших args элемент с индексом i возводить его в степень n и прибавлять к нашей переменной s. Далее сделаем так: у нас будет вывод текста, к нему прибавим двоеточие, чтобы оно прилипло к сумме чисел и выведем, соответственно, сумму. Получилась такая вот незамысловатая функция(рис.17).

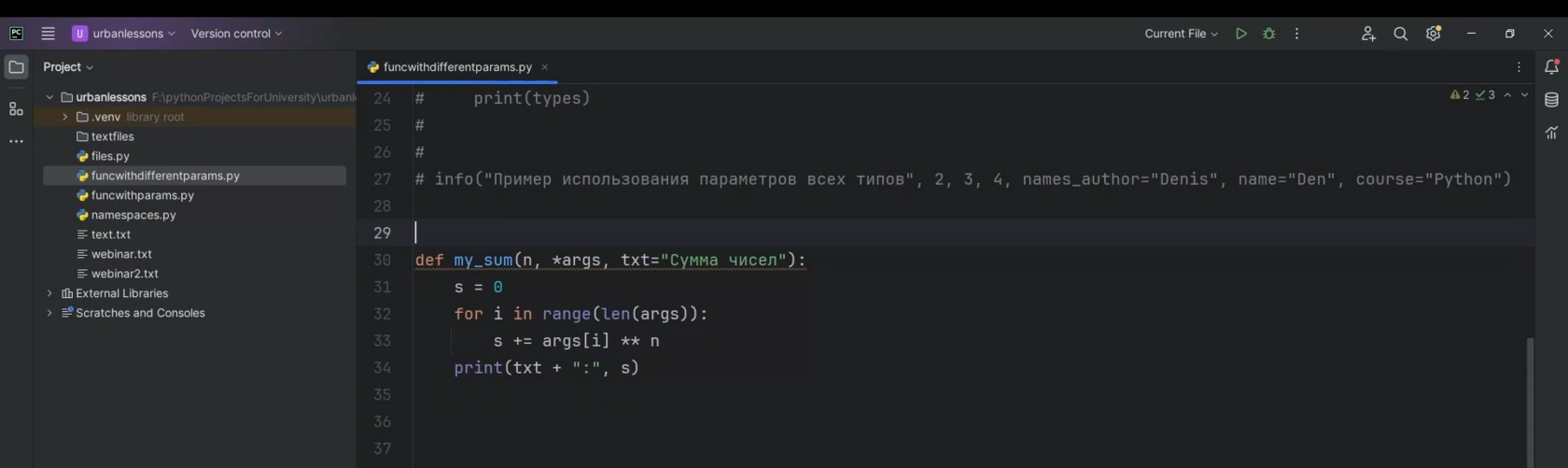


Рис.17

Если мы будем ее вызывать и передам изначально единицу как степень, первый позиционный параметр, все остальные уже будут соответственно попадать в кортеж. При запуске вышло “Сумма чисел: 15”(рис.18). Что произошло?

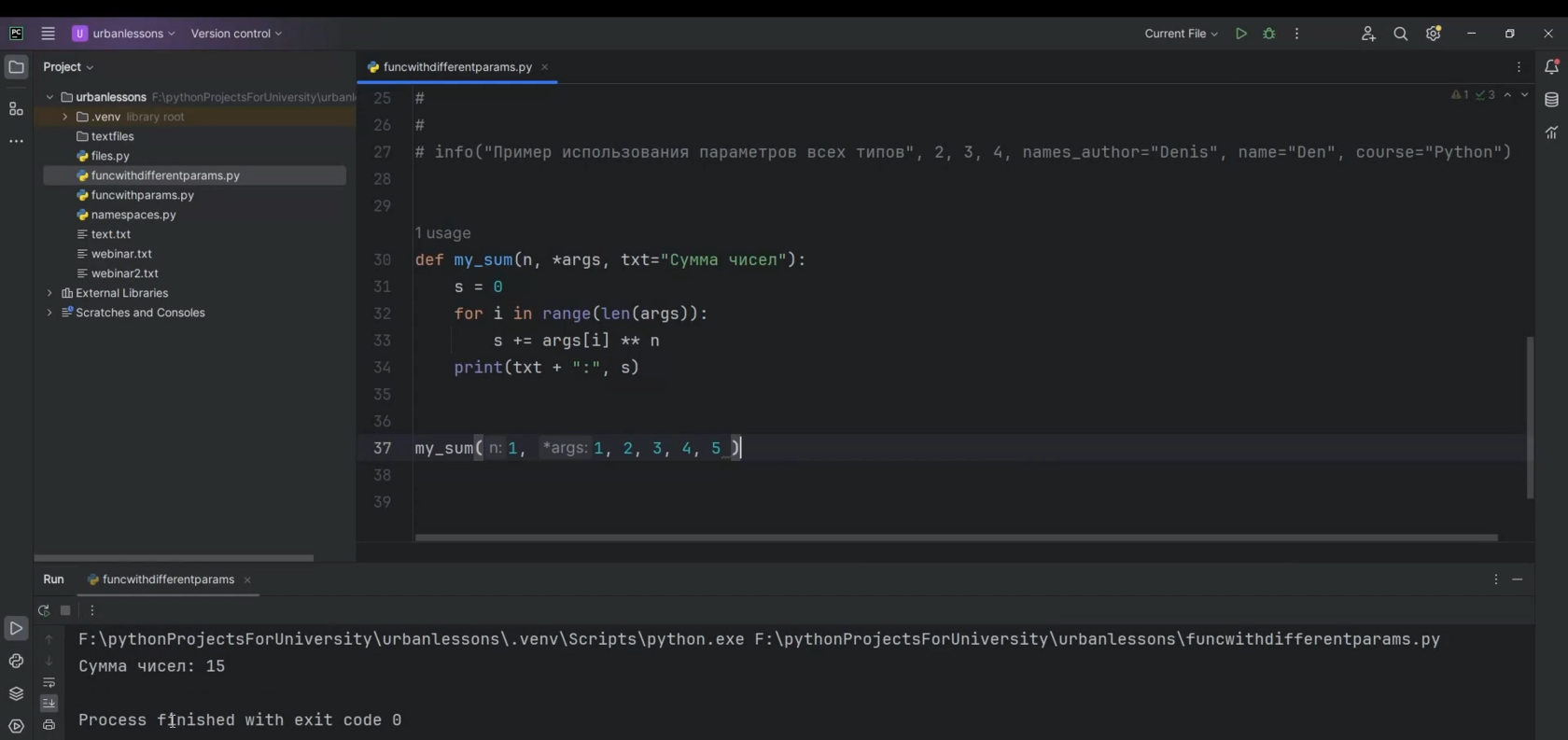


Рис.18

Первый параметр улетел на место позиционного параметра n, то есть первый аргумент, если быть точнее. Те последующие аргументы, которые мы передавали, они улетели в параметр args, то есть в виде кортежа. Затем цикл for рассчитанный на количество повторений, равный на количество элементов нашего кортежа, в котором мы берем элемент по его индексу: начинаем с элемента 0, 1, 2, 3 и так далее, до тех пор, пока мы не переберём весь кортеж. Берем этот элемент возводим его в степень, которую мы здесь написали и прибавляем к переменной s. В конечном счете получится какая-то сумма.

Если нам нужна сумма квадратов, мы снова вызовем my\_sum. Степень будет 2, а в args напишем 2, 3, 4, 5 точно так же, но параметр txt изменим не на сумму чисел, а на сумму квадратов. Запустим. Сумма чисел 15, сумма квадратов 54(рис.19). То есть получилась такая универсальная функция. Можем то же самое повторить, например, сделать для суммы кубов, суммы 4 степеней и так далее. Здесь комбинированная применяется и обычная, позиционный параметр и переменное количество позиционных параметров и именованный параметр.

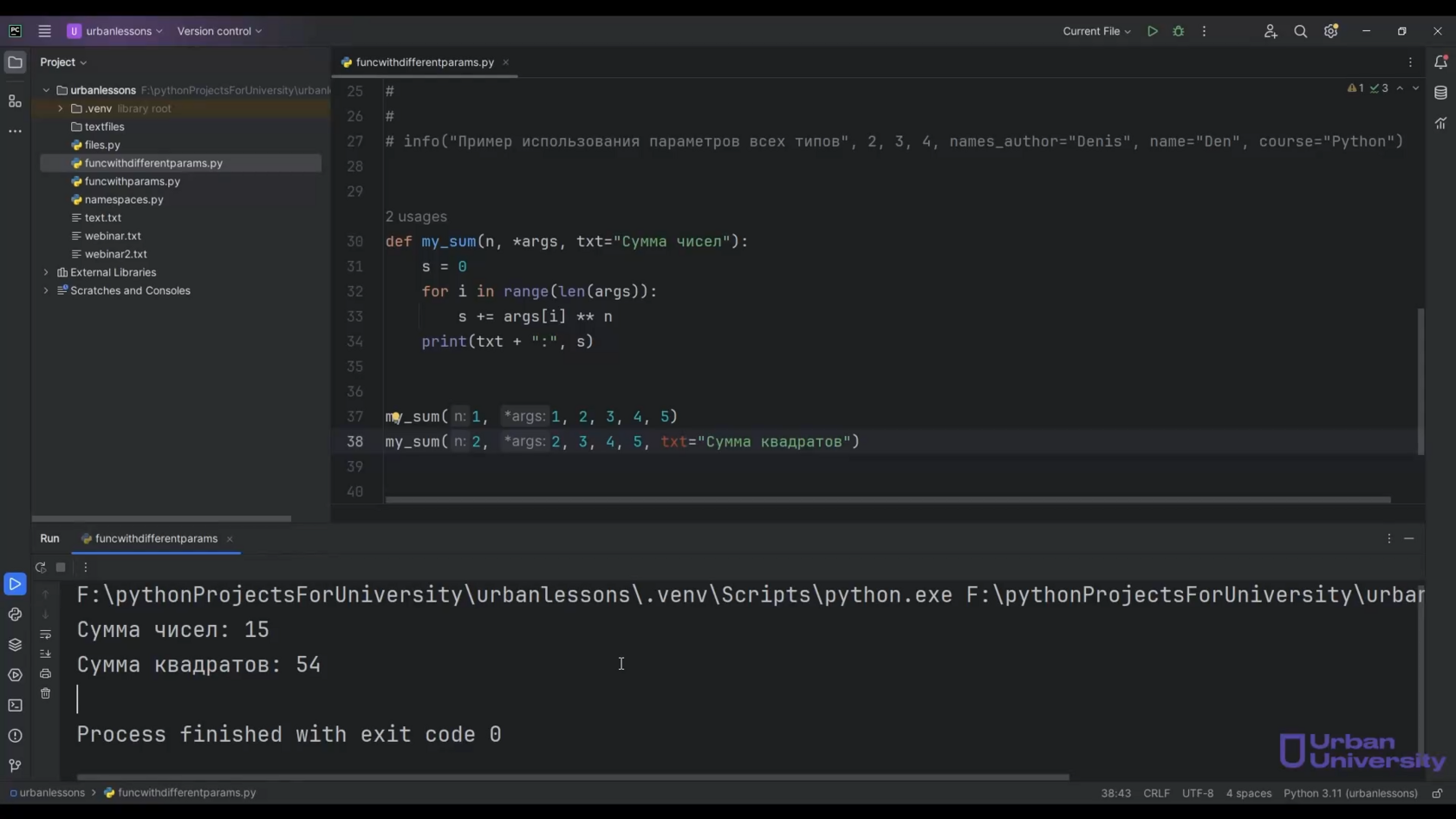


Рис.19

При необходимости, если у кого-то есть в этом потребность, сюда можно добавить ещё и произвольное количество именованных параметров. То есть таким образом мы подстраховываемся от тех случаев, от тех ситуаций, когда мы не знаем, сколько нам необходимо будет передать аргументов функции, сколько пользователь передаст или какое количество этих аргументов будет использоваться в вычислениях. В этих случаях мы можем использовать либо одну “звездочку” для позиционных параметров, для произвольного количества и 2 “звездочки” для произвольного количества именованных параметров.