**2023/11/11 00:00|Лекция. Введение. Строки байты и кодировки**

## Введение. Строки байты и кодировки

На сегодняшнем занятии мы начинаем изучение новой для нас темы. Ранее мы привыкли работать с вами с обычными строками. Однако в этом разделе мы будем работать с файлами и можем столкнуться с некоторыми проблемами, которые могут возникать при чтении этих файлов. Для того чтобы быть готовым к решению этих проблем, нам с вами нужно немного углубиться в то, как устроена работа с этими самыми строками.

Мы привыкли видеть строки обычного формата. То есть, если мы напишем строку «hello», для нас она и будет строкой «hello». При выводе мы точно также получим строку «hello» (Рис.1).

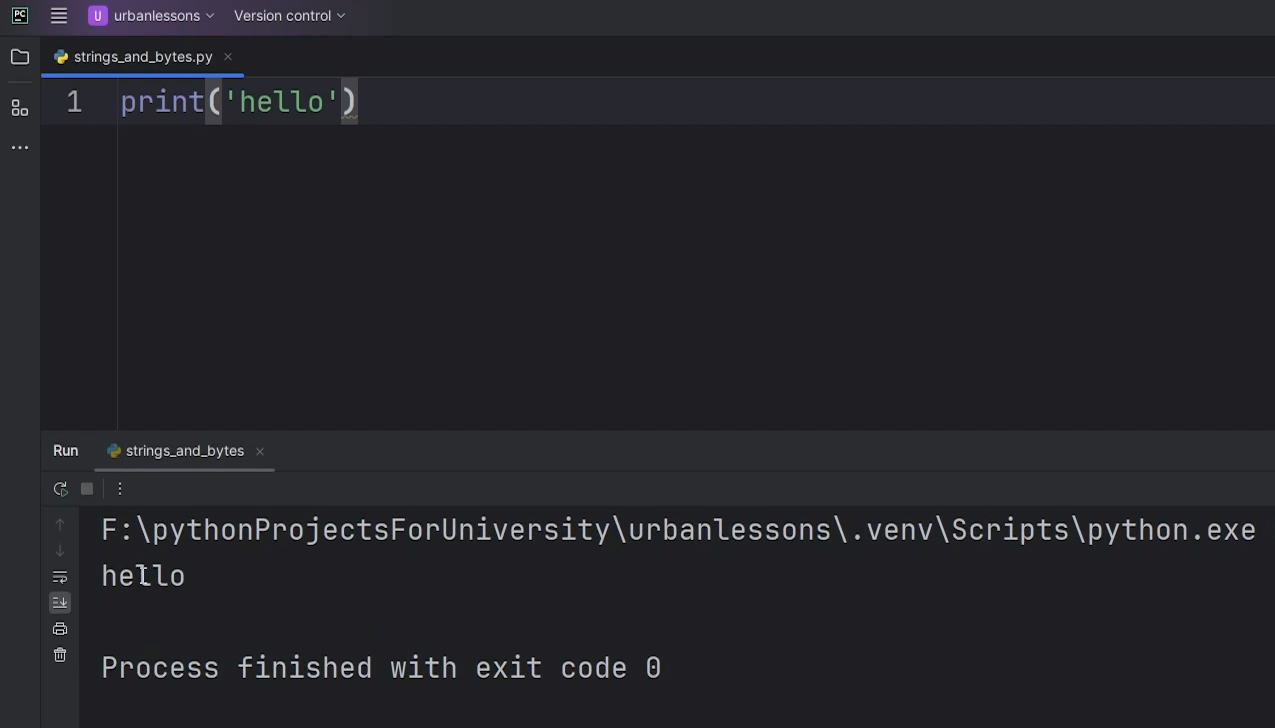


Рис.1

Однако компьютер изначально не понимает эти символы. Для каждого символа есть соответствующее число, по которому у нас компьютер будет понимать, что речь идёт именно об этом символе. Это так называемая **кодировка.** Существует некая **таблица**, которая называется **«ASCII»**. Это таблица, в которой содержатся символы, и есть определённая связь у каждого символа и у определённого числа. Она содержит в себе символы от 0 до 127.

Если мы попробуем взять отдельный символ для того, чтобы вывести число для определённого символа, мы с вами можем воспользоваться такой функцией встроенной, которая называется «ord». Если мы, допустим, запустим, выведем численное представление для символа «а», мы получим 97 (Рис.2). Для заглавной «А» мы получим 65 (Рис.3).

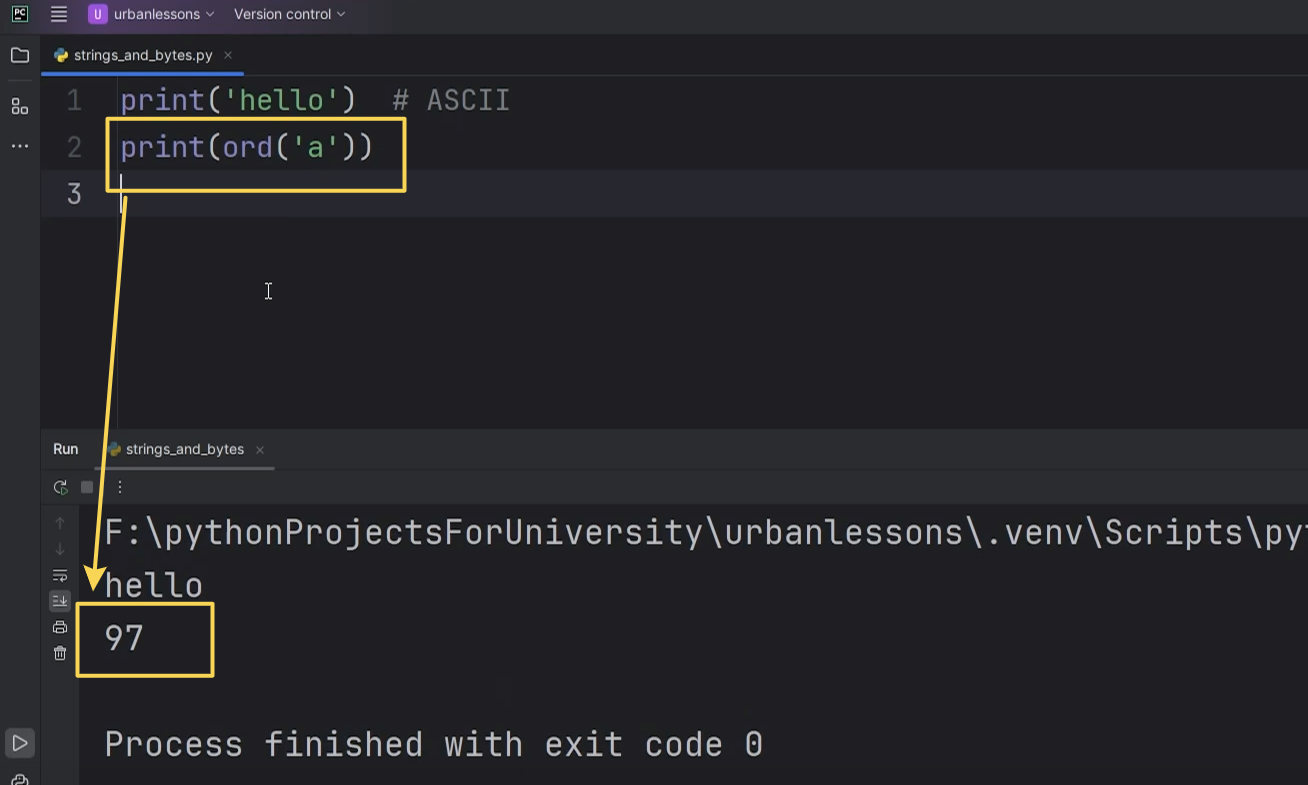


Рис.2

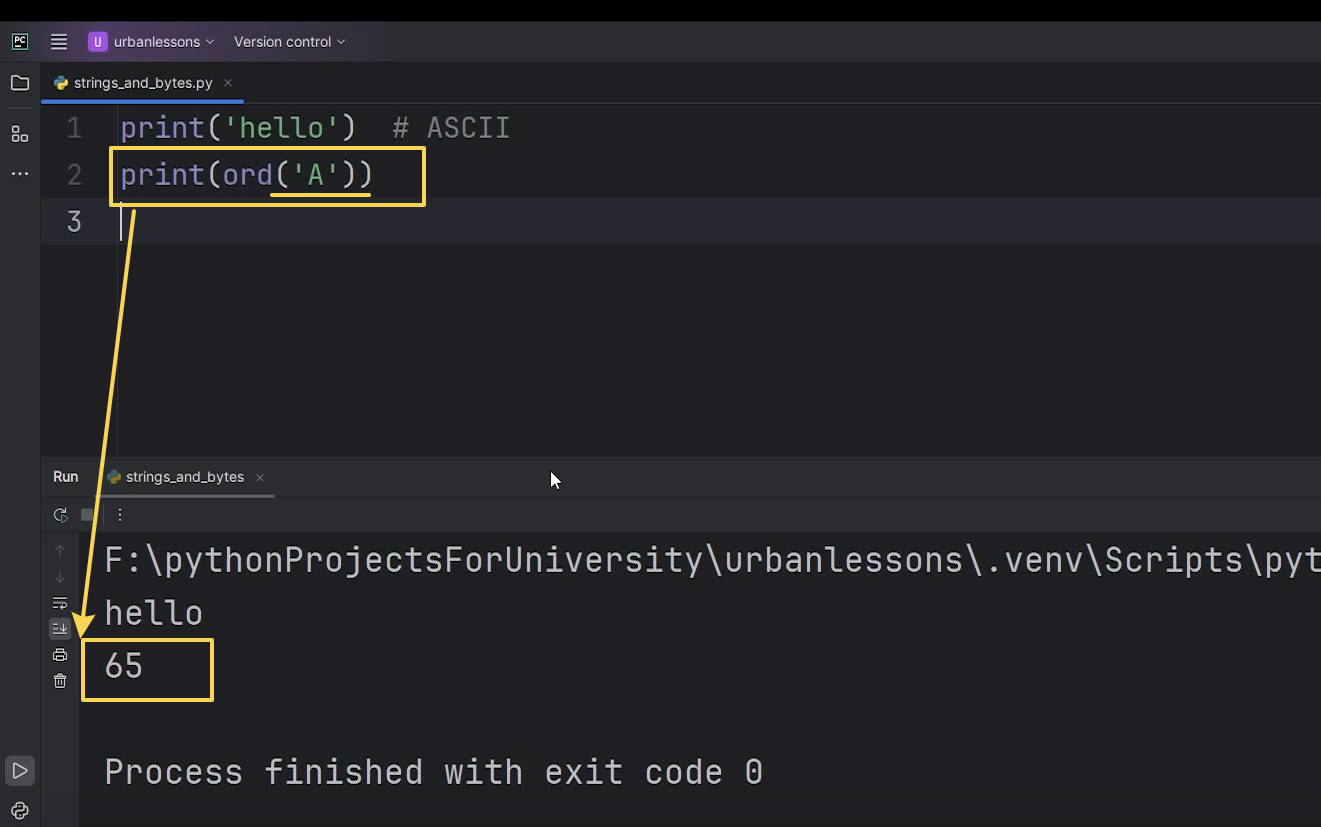


Рис.3

У нас символы не стоят друг с другом. Однако смотрите, как это работает? У нас есть символ «h» -104 (Рис.4). У «е», соответственно, будет свой символ, это уже 101 (Рис.5).

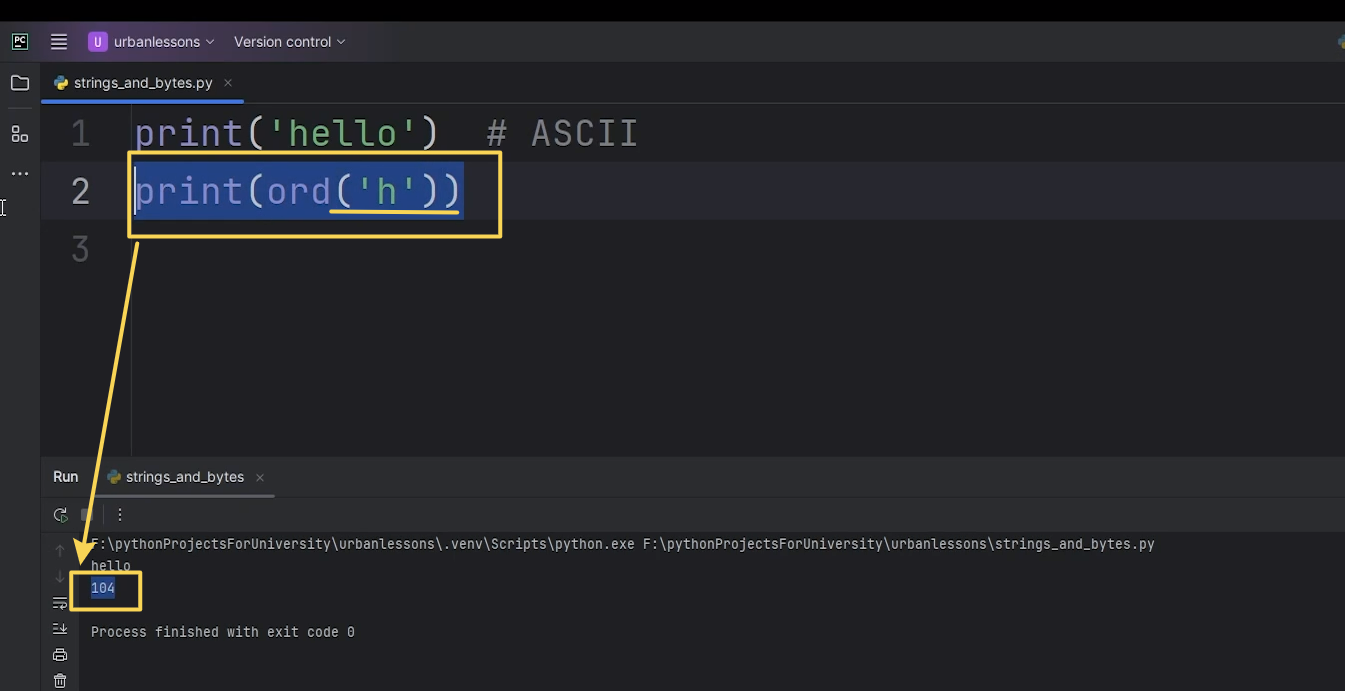


Рис.4

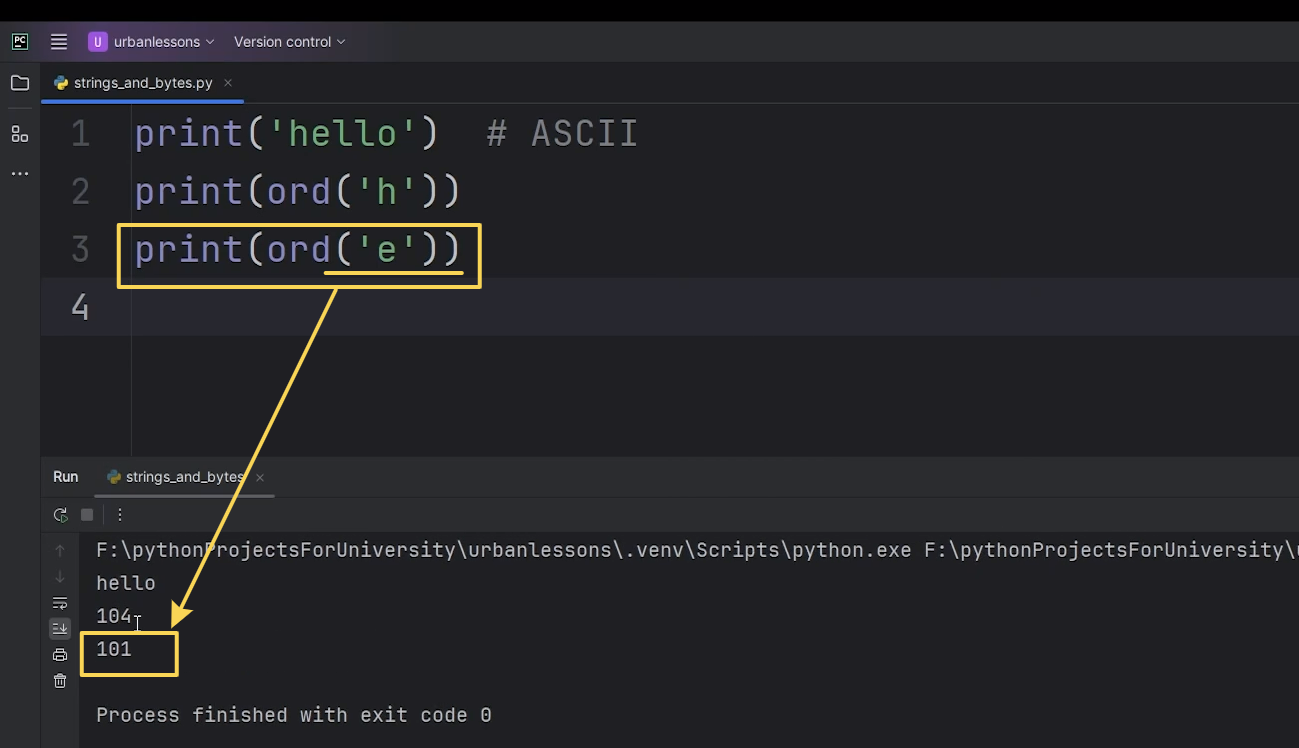


Рис.5

Давайте всю строку «hello» мы переведём в эти числовые значения, связанные с этими символами. Для этого «hello» мы с вами сохраним в какую-нибудь переменную, например, «a». И циклом пройдёмся по этой переменной. Создадим список, назовём его «chars». Он пока будет пустым. И далее мы, перебирая каждый символ в нашей строке, будем сохранять его представление в виде числа и добавлять в этот список «chars». То есть, получается, мы будем использовать функцию «ord» для того, чтобы перевести символ в число. Будем брать каждый отдельный символ нашей строки и добавлять его в список «chars», используя метод «append», уже знакомый нам. И на выходе давайте посмотрим на этот список «chars» (Рис.6).

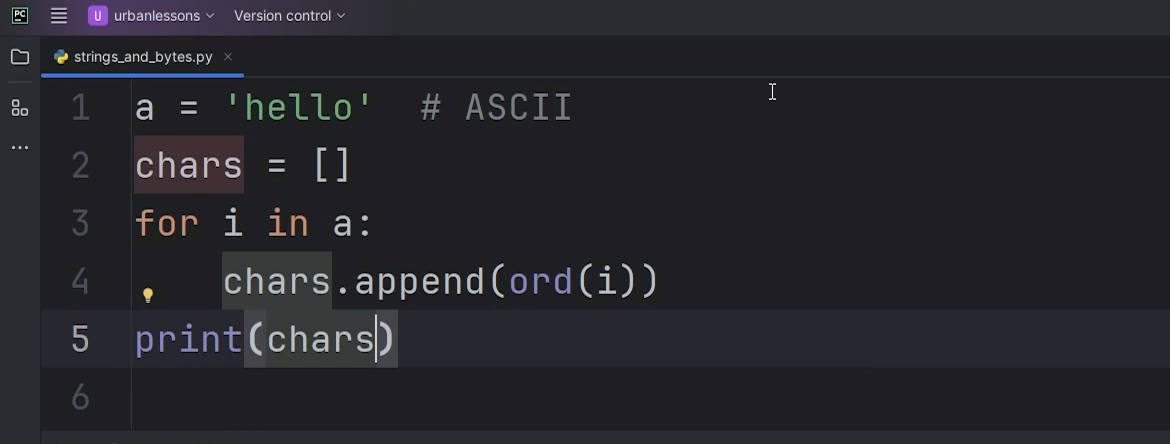


Рис.6

Мы получим соответствующие номера каждого символа в нашей строке (Рис.7).

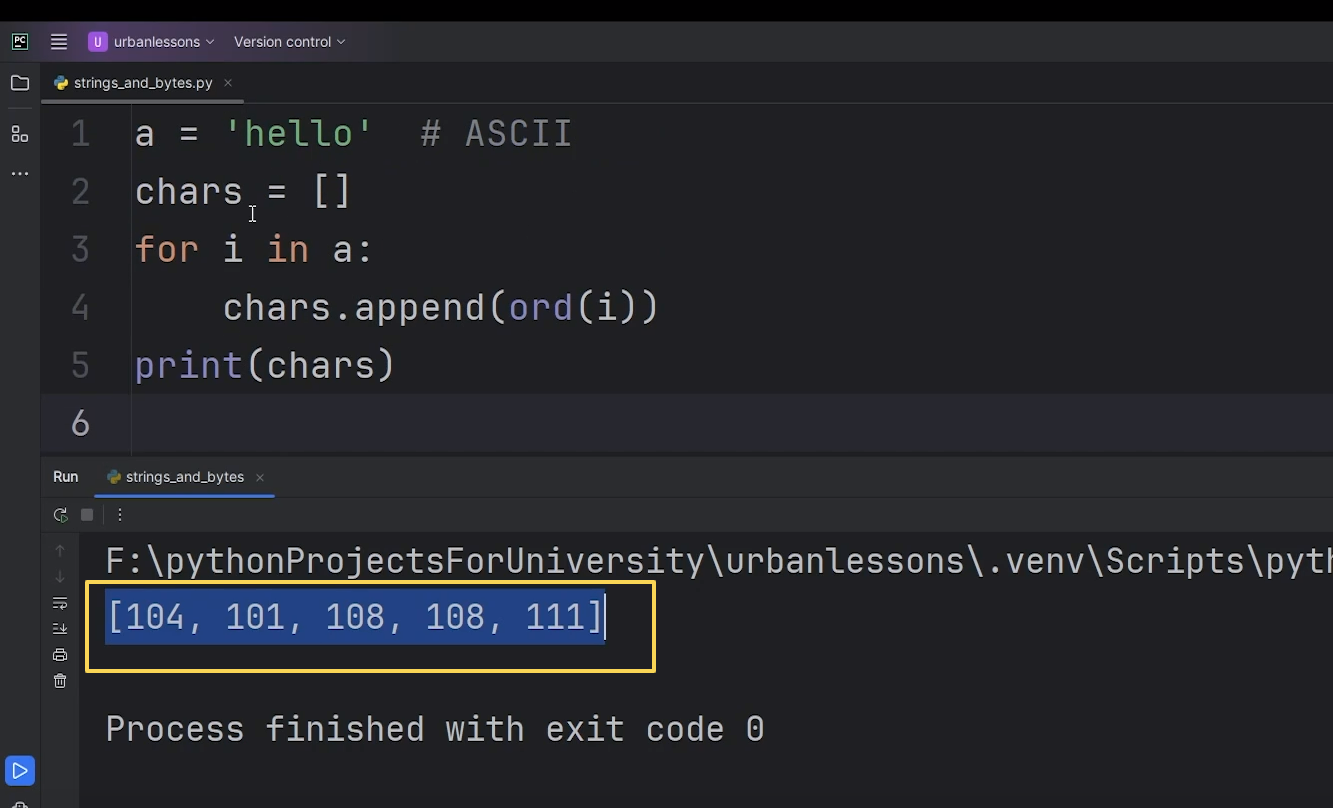


Рис.7

А теперь попробуем выполнить обратное преобразование. Для этого создадим переменную «s». Пусть это будет пустая строка. И точно также циклом «for» мы будем пробегаться по нашему списку из этих символов. Чтобы перевести число обратно в соответствующий символ, нам нужно использовать функцию «chr», которая тоже является встроенной. И мы будем перебирать каждое число. Первое переводить его в соответствующий символ и прибавлять к пустой строке. В результате мы получим полное слово. Давайте выведем строку и запустим (Рис.8). У нас получается уже строка «hello».

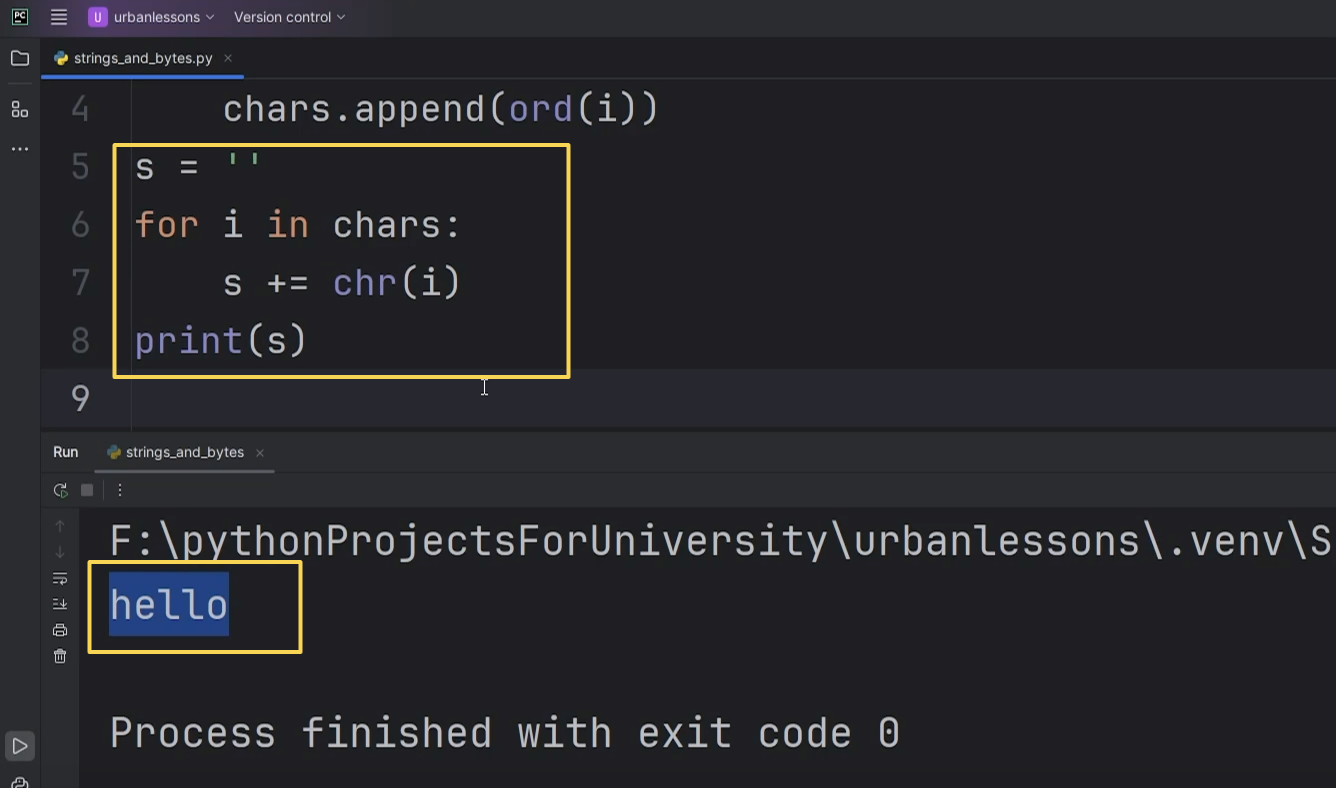


Рис.8

Таким образом, есть связь с абсолютно каждым символом, который мы с вами привыкли использовать. Предыдущий код закомментируем. Если мы запустим цикл на 128 повторений и будем выводить представление этих чисел, скажем так, в виде символов, мы получим определённые значения (Рис.9, Рис.10). Некоторые видите у нас ещё не читаются, но дальше идут более знакомые для нас знаки, которые мы используем при написании нашего кода, включая цифры, заглавные и маленькие буквы.

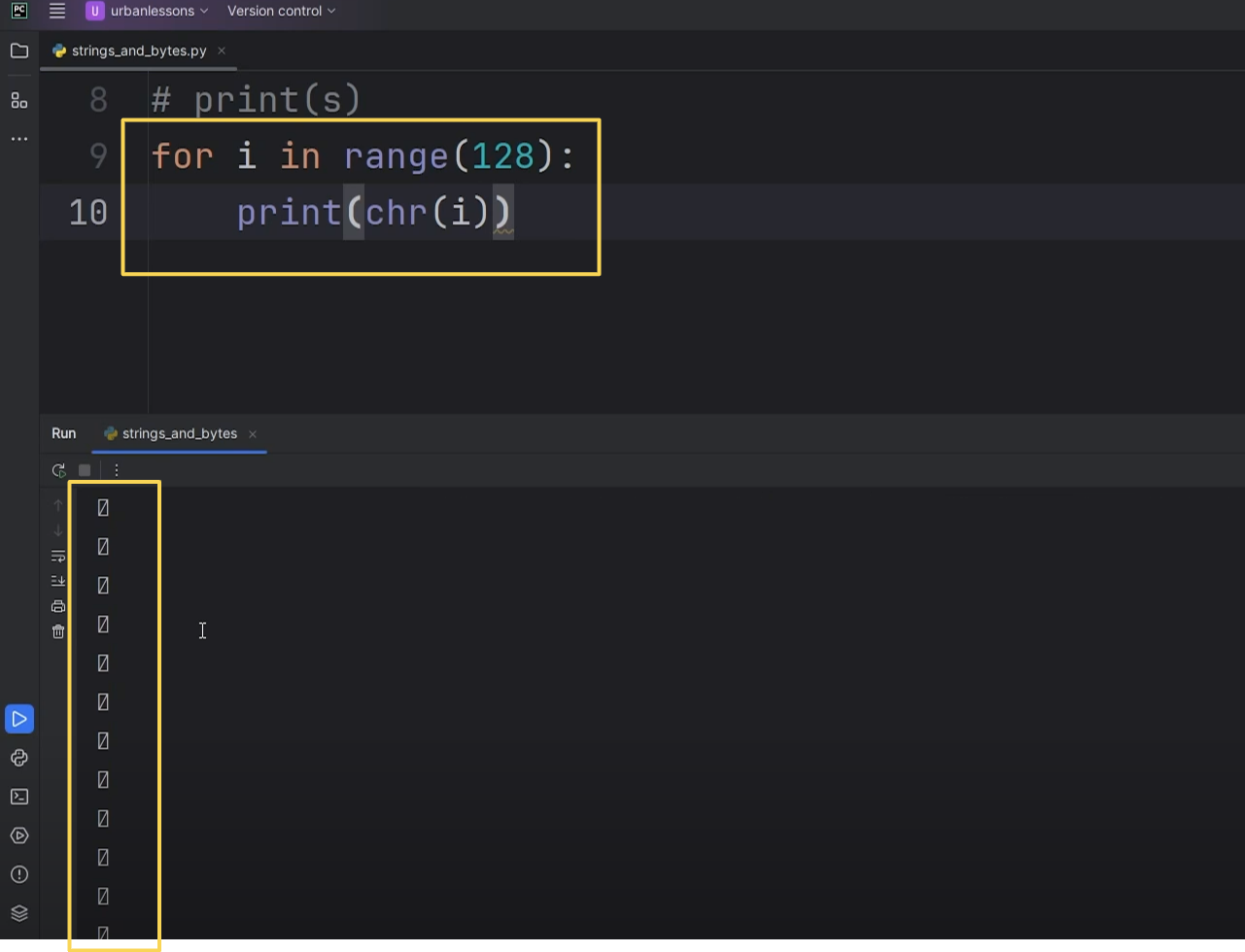


Рис.9

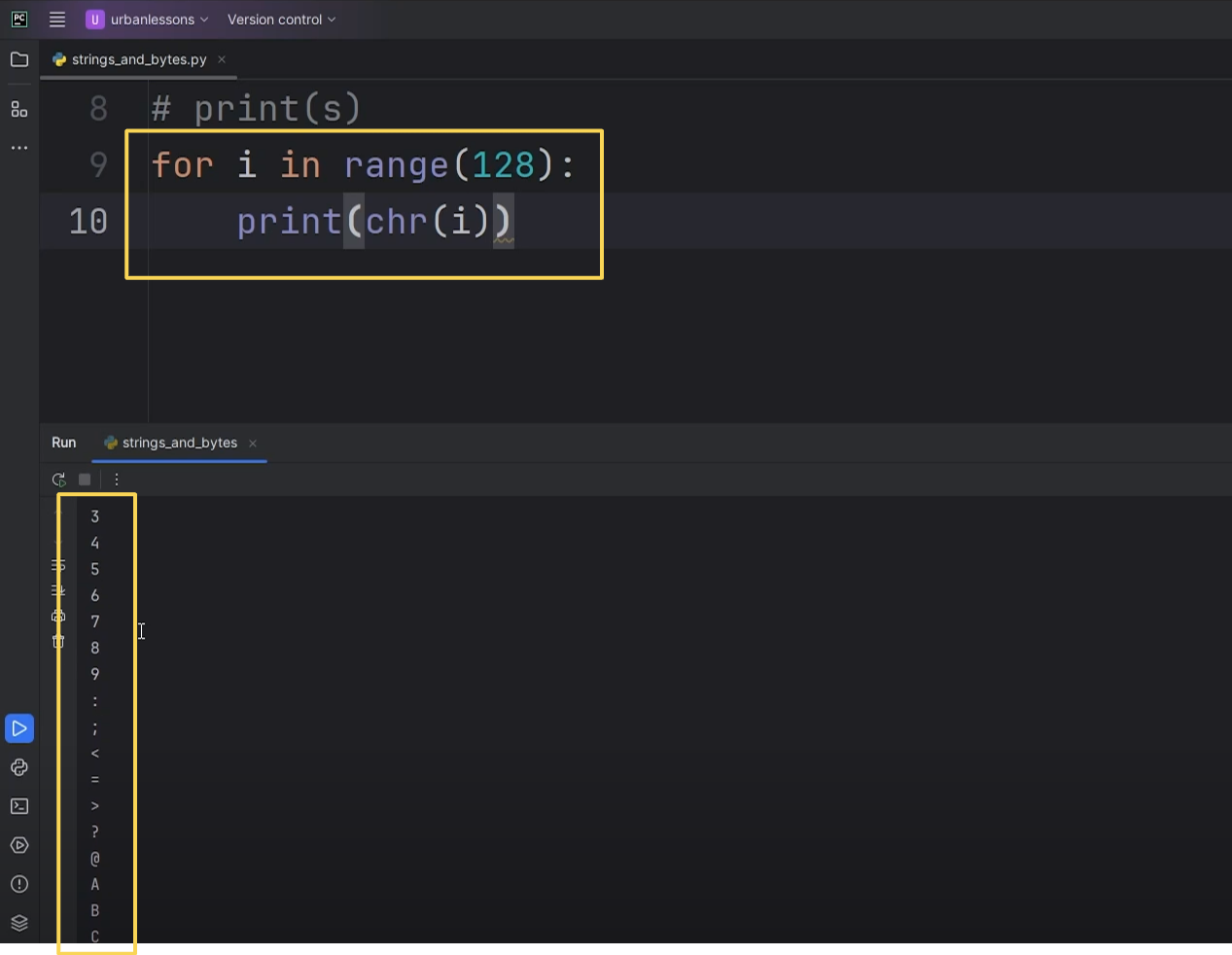


Рис.10

По сути, эти символы из таблицы «ASCII». Однако эта таблица содержит всего лишь 127 символов, которые используются у нас наиболее часто. Но Python больше придерживается таблицы «Unicode». Она очень похожа на «ASCII», однако содержит в себе более 2000000 символов. То есть, если «ASCII» содержит в себе только лишь 127, точнее, 128 получается у нас, то «Unicode» содержит гораздо больше символов. Но принцип работы точно такой же: есть определённое значение, которое связано с определённым символом.

Мы можем взять, например, от 1000 до 1200, и вывести соответствующие символы этим числам (Рис.11, Рис.12). Можно увидеть уже незнакомые для нас. Там идут русские буквы, которые, конечно, мы с вами прекрасно знаем, но остальные мы не видели.

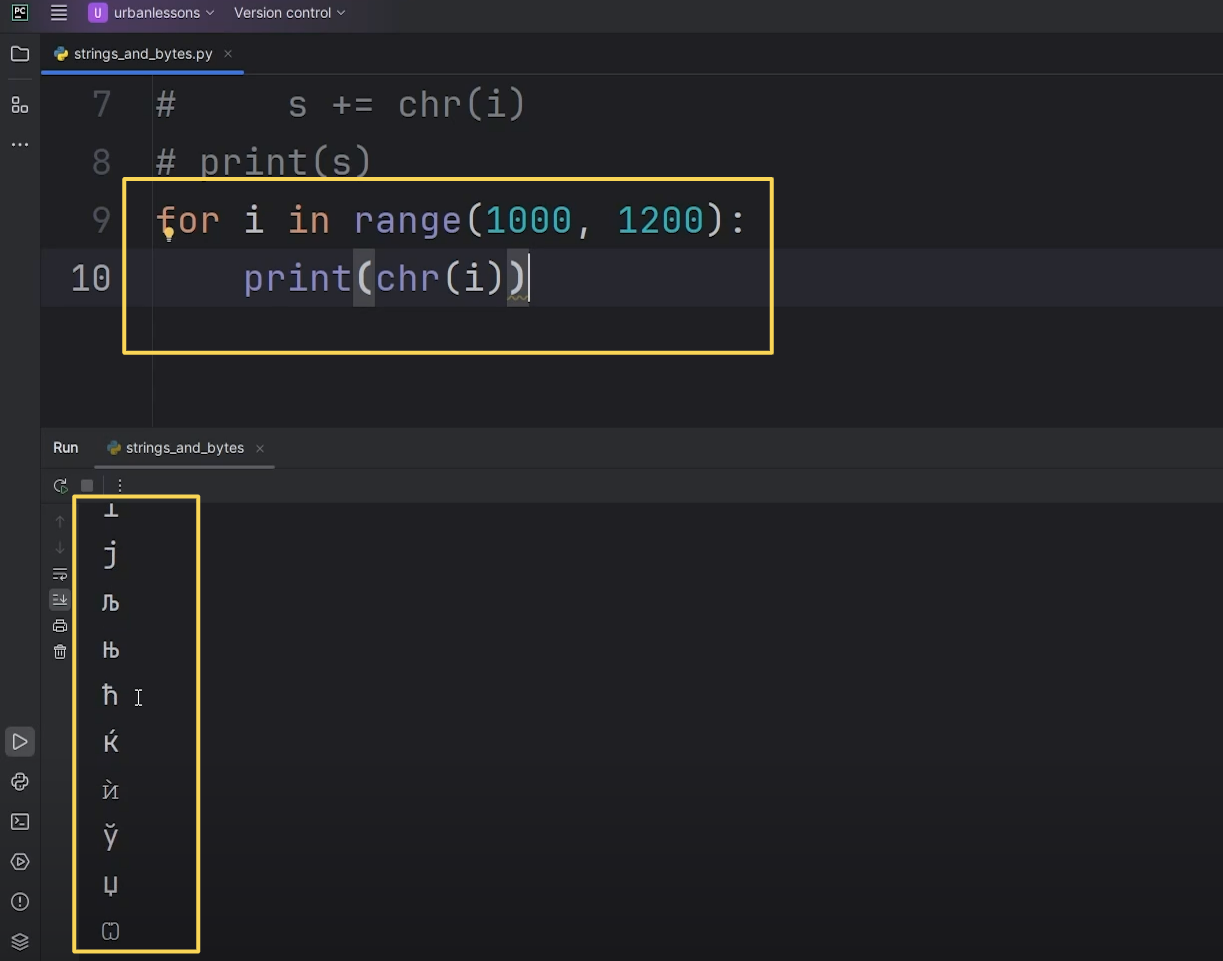


Рис.11

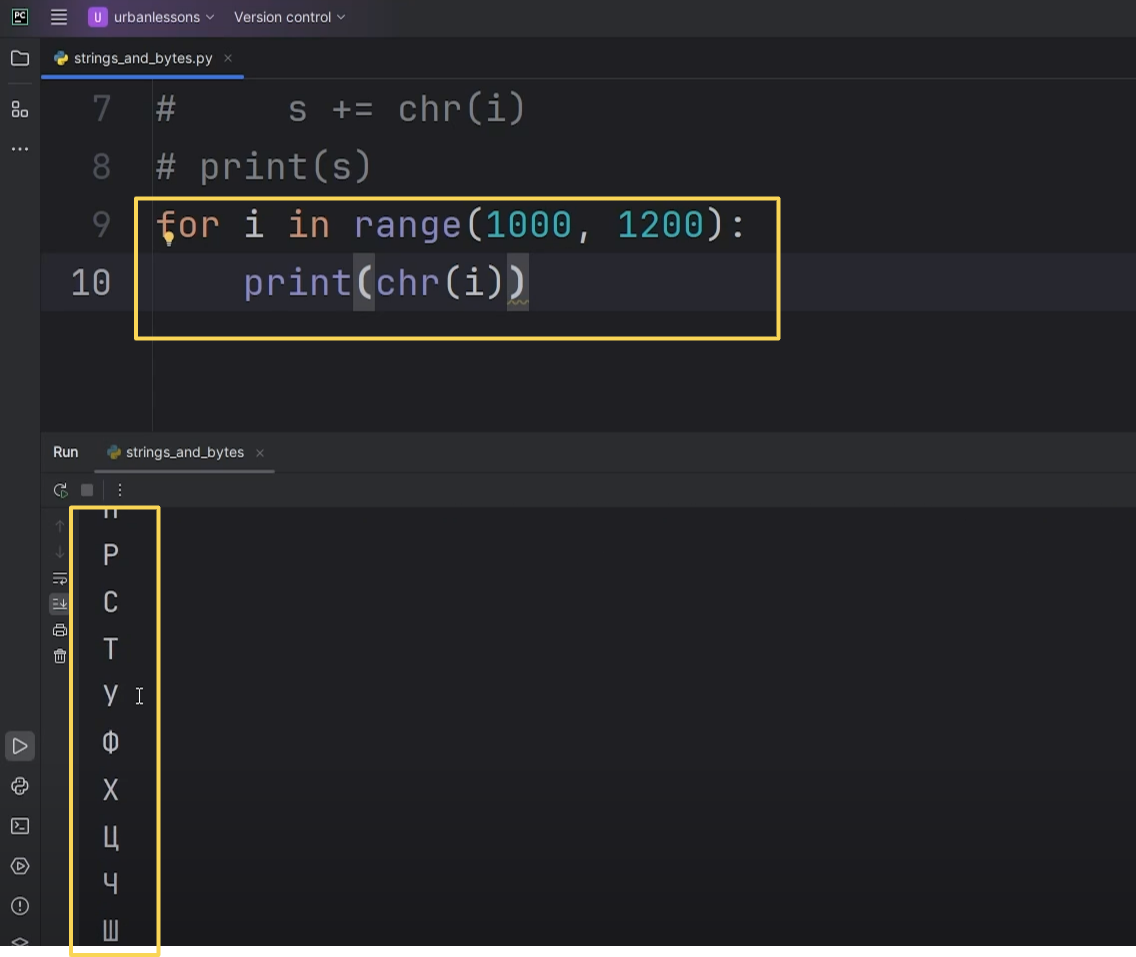


Рис.12

Это подтверждение того, что Python работает именно с таблицей «Unicode», которая содержит в себе гораздо больше символов. Туда входят и символы разных алфавитов, например, китайский, английский, тот же русский. В общем, очень большое их количество.

Когда мы с вами считываем какую-то информацию, компьютер тоже выполняет определённые действия. Правда, он выполняет эти действия уже на уровне интерпретатора. То есть нас не заботит о том, как эти символы будут переводиться, что с ними будет происходить. Мы просто с ними работаем в той или иной манере. Однако на уровне интерпретатора компьютеру проще это понимать, если он работает с байтами.

**Байты**-это представление строкового какого-то символа или строки целиком в двоичном или шестнадцатеричном представлении. Для этого существует даже отдельный тип данных для работы с байтами. И байты у нас выглядят как обычная строка, однако с префиксом «b» перед строкой. После префикса может быть что угодно. Например, давайте переведём с помощью функции «hex», воспользуемся ещё функцией «ord», то есть, чтобы перевести из символа «h» его в числовое представление и это представление мы с вами переведём в шестнадцатеричный формат (Рис.13).

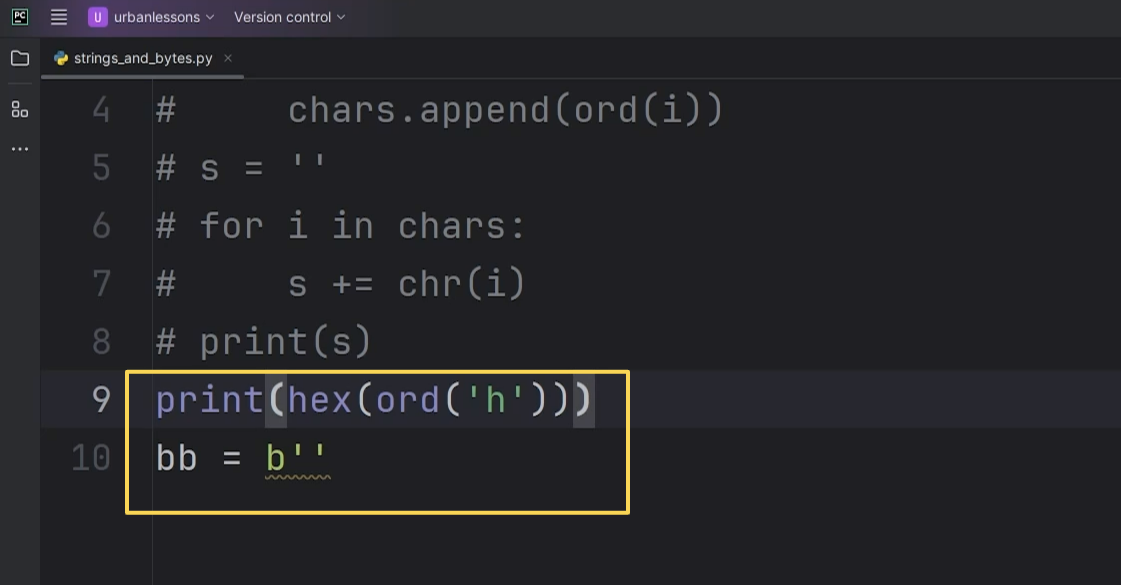


Рис.13

Запустим и получим «0x68» (Рис.14).

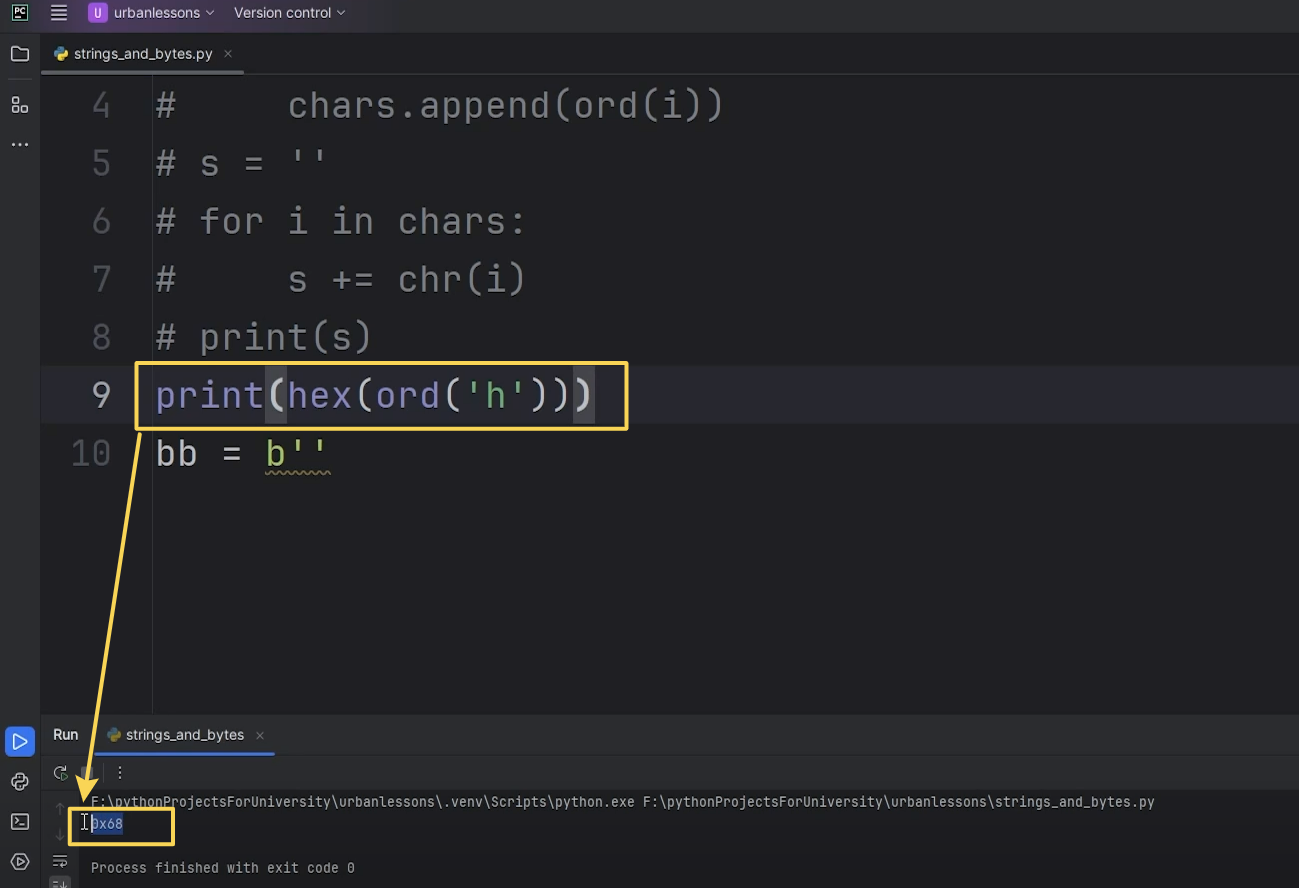


Рис.14

Изменим строку вот таким вот образом (Рис.15).

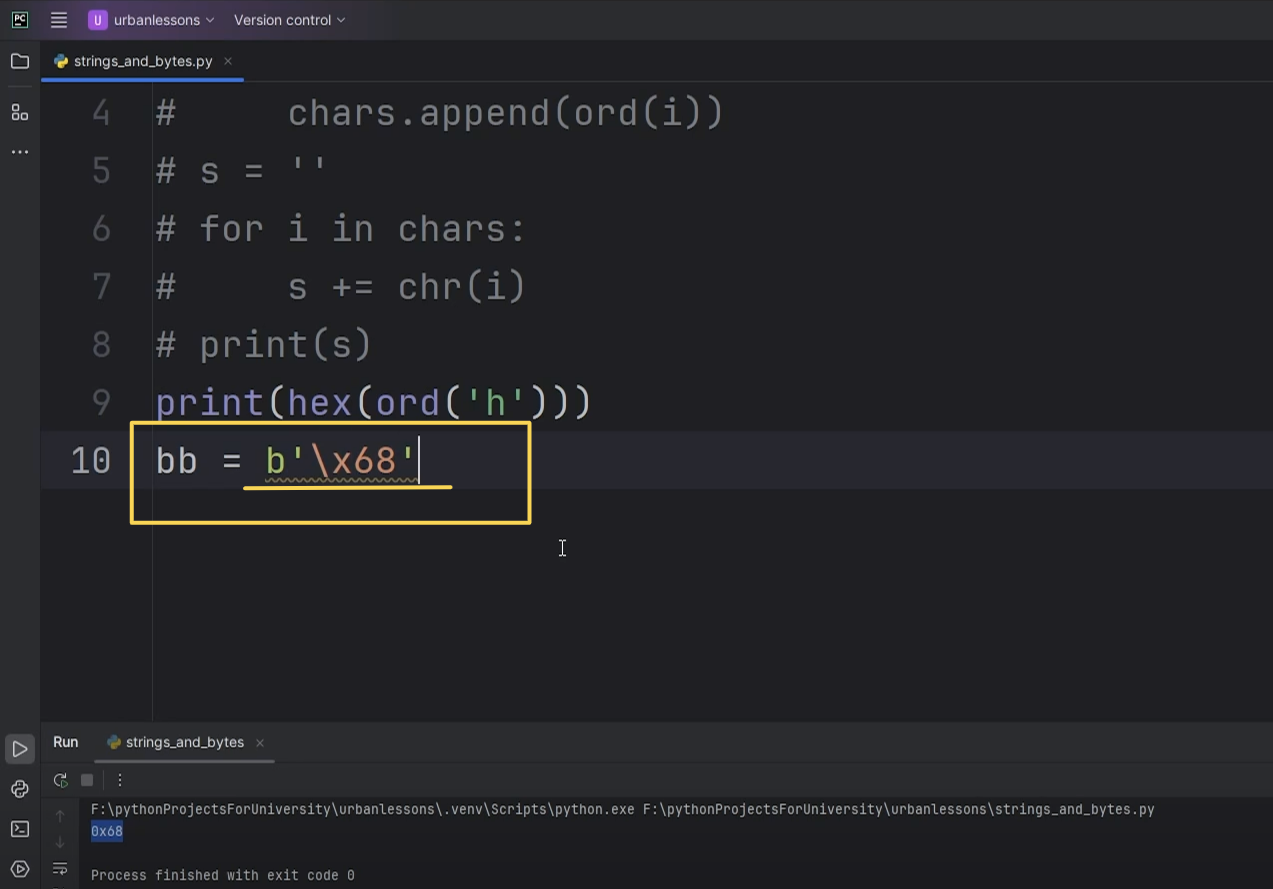


Рис.15

Если мы выведем тип у этой строки, то мы видим то, что эта строка уже не принадлежит к «str», а относится к классу «bytes», именно к байтам (Рис.16).

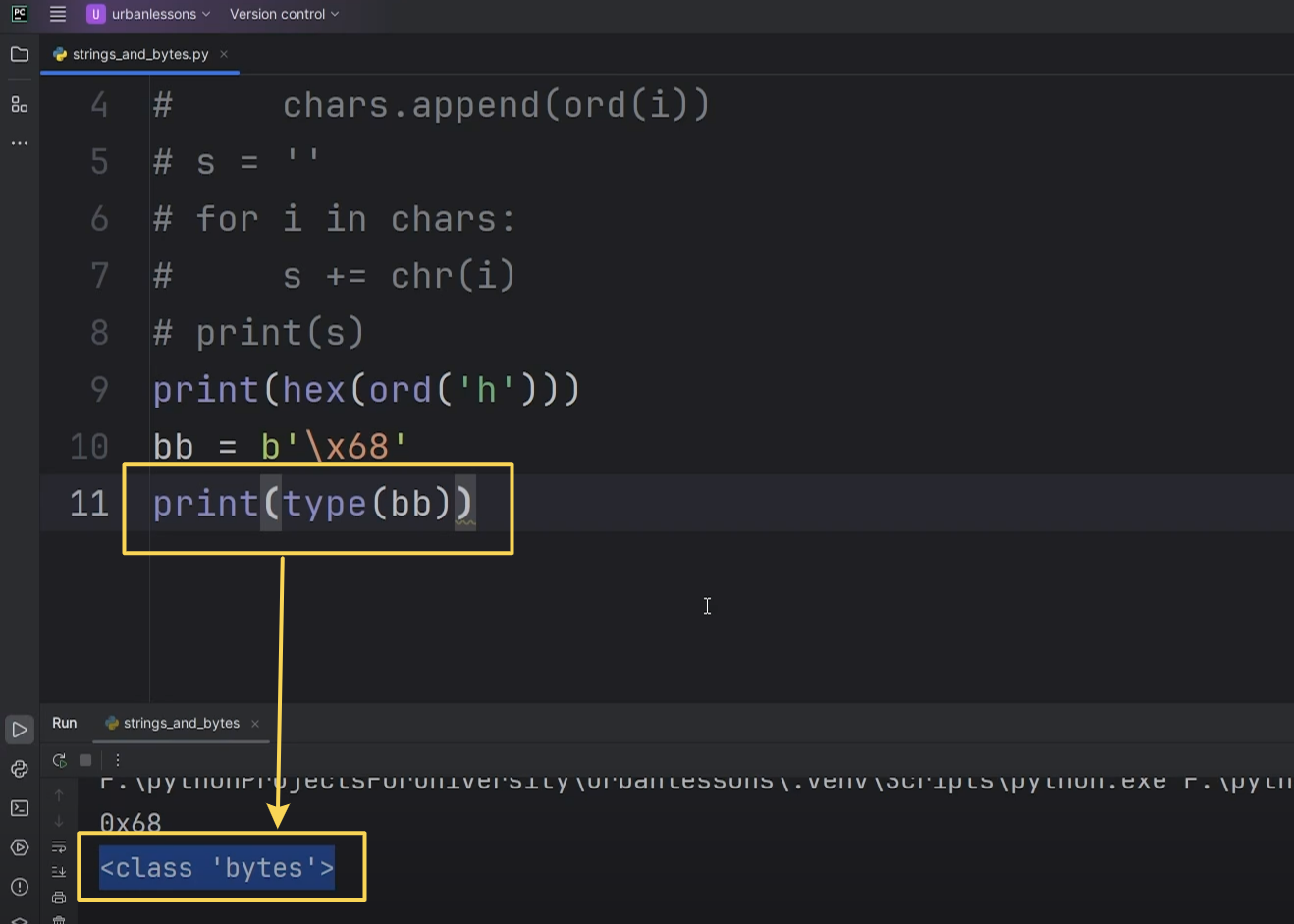


Рис.16

Теперь воспользуемся функцией «decode». И уже на выходе мы с вами получим «h», тот самый символ (Рис.17).

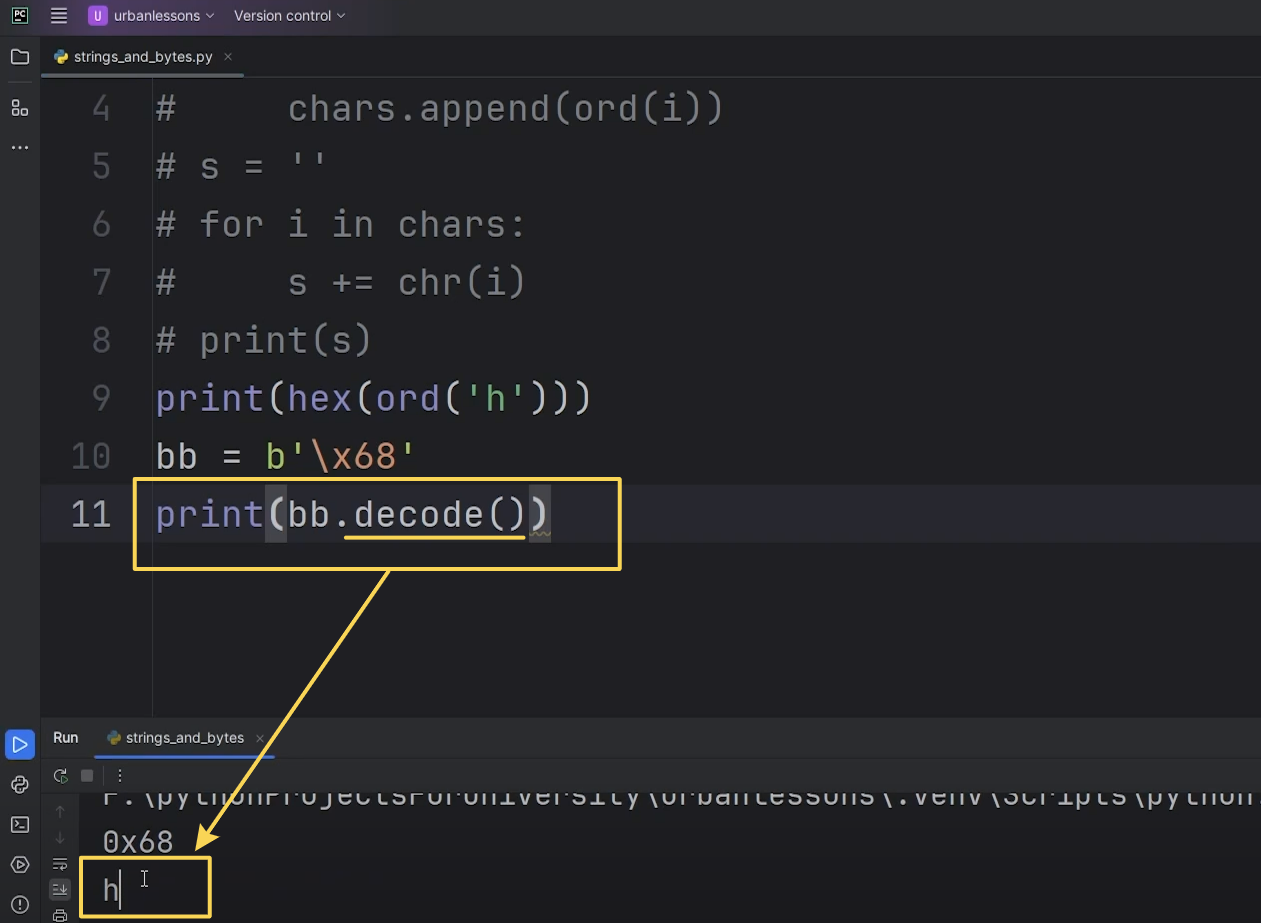


Рис.17

Интерпретатор сам выполняет эти операции по кодированию и декодированию непосредственно символов из байт-кодов в то, что мы с вами привыкли делать. Интерпретатору, конечно, проще работать с байтами.

Если уж углубляться в эту тему, то на самом деле при работе с байтами у нас есть возможность передавать не только строковые значения, либо как-то хранить их в памяти. Но есть также возможность и хранить, и передавать мультимедийные значения, то есть видео, фотографии. Базы данных часто представляют какие-то отдельные элементы в виде байтовых последовательностей. Однако в эту тему углубляться нам сейчас, конечно же, не стоит. Просто мы с вами должны понимать, что существует и такая история как байты. По сути кодирование и декодирование у нас выполняет интерпретатор самостоятельно. Однако мы можем и вручную, как в примере выше, например, раскодировать байтовую строку, привести её вот к соответствующему символу. Так и строку мы можем перевести в последовательность байтов.

Для того чтобы нам было комфортно работать с файлами в этом модуле, нам просто нужно ознакомиться или понимать такую вещь, как кодировка, про которую мы вам говорили в самом начале данного урока. То есть это соответствие определённого символа определённому числовому значению. Иногда мы можем при работе с файлами встречать набор из непонятных для нас символов. То есть, если мы пытаемся считать информацию из файла, получаем набор непонятных значений нечитабельных, значит, вероятно, произошла ошибка или непонимание при кодировке. Их на самом деле большое количество наиболее распространённый, общепринятый стандарт кодировки это «UTF-8». Там есть ограничения по тому, сколько байт занимает места в памяти каждый символ непосредственно того, что мы передаём. Однако это самый распространённый формат. С такими проблемами связанными с кодировкой мы с вами столкнёмся и подробнее их рассмотрим, когда уже научимся работать с файлами.