**Методический материал по курсу**

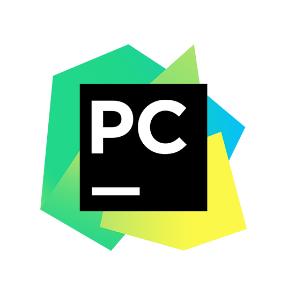
«16199 Оператор электронно-вычислительных и вычислительных машин с основами языка программирования Python»

**Тема 1.1** **Знакомство с Python 3.9 и программным обеспечением PyCharm Community Edition 2022**

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

PyCharm — это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик.

Для начала работы с PyCharm, необходимо скачать его с официального сайта <https://www.jetbrains.com/ru-ru/pycharm/>. После на рабочем столе появится иконка.



Создадим проект, для этого необходимо:

1. Запустить PyCharm;

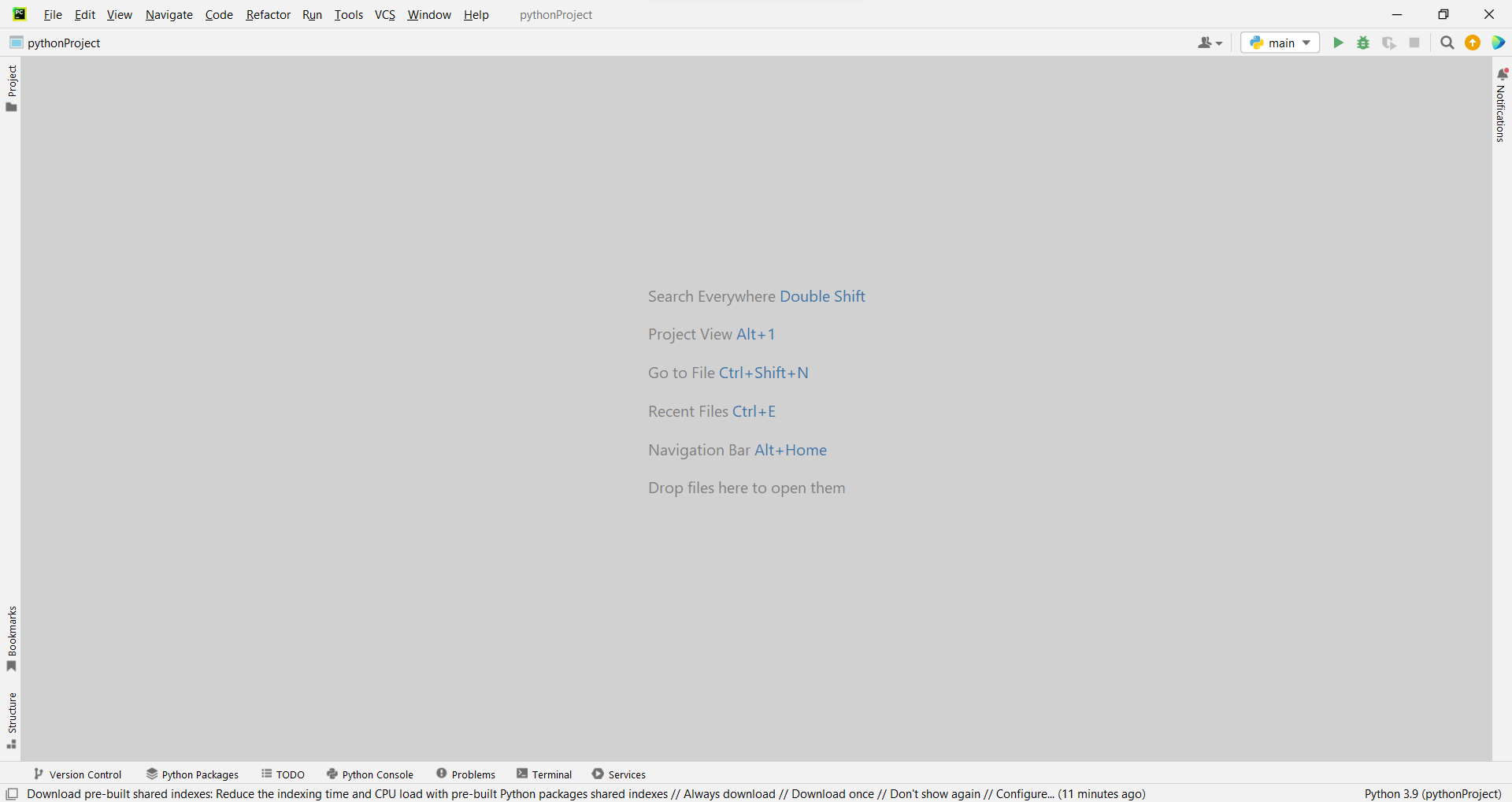
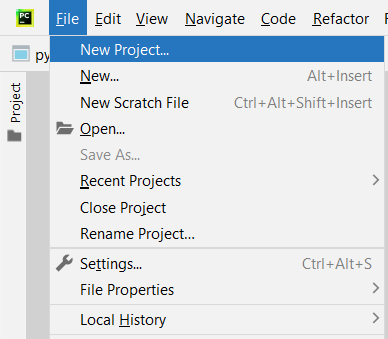


Рисунок 2

1. Нажать в верхней панели File – New Project;

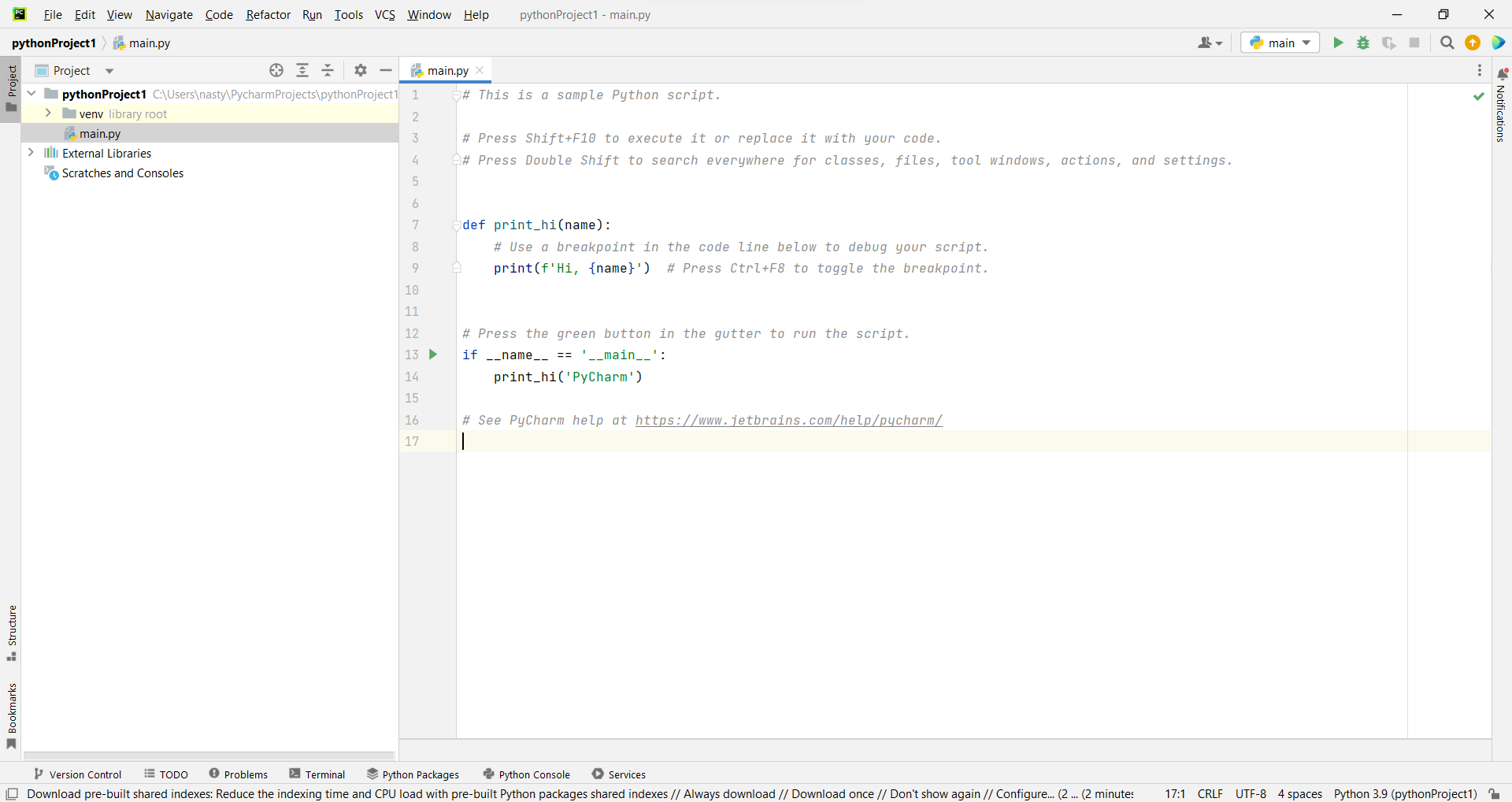


1. Выбрать местоположение вашего проекта и назвать его (обычно стоит по умолчанию);



1. Нажать кнопку Create.

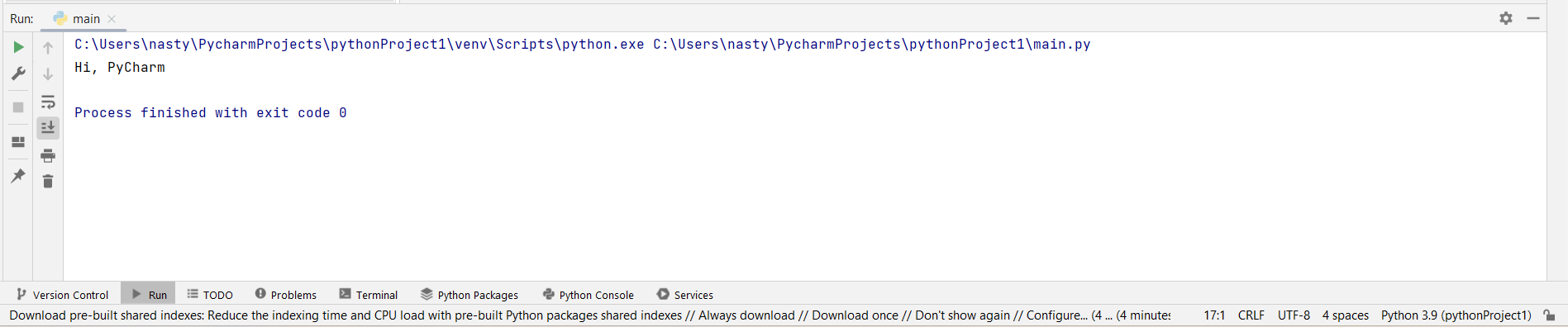
В итоге должна открыться рабочая область.



Изначально, после создания проекта, стандартный код пишется по умолчанию. Для того, чтобы запустить проект, нажмем Run



По завершению в консоль, которая находится снизу, будет выведен результат. На данный момент это фраза «Hi, PyCharm».



**Задание по теме 1.1 Знакомство с Python 3.9 и программным обеспечением PyCharm Community Edition 2022**

Создайте папку на рабочем столе, наименование должно содержать группу и фамилию:



Создайте проект в созданной папке, название должно содержать вашу фамилию. Попробуйте запустить проект.

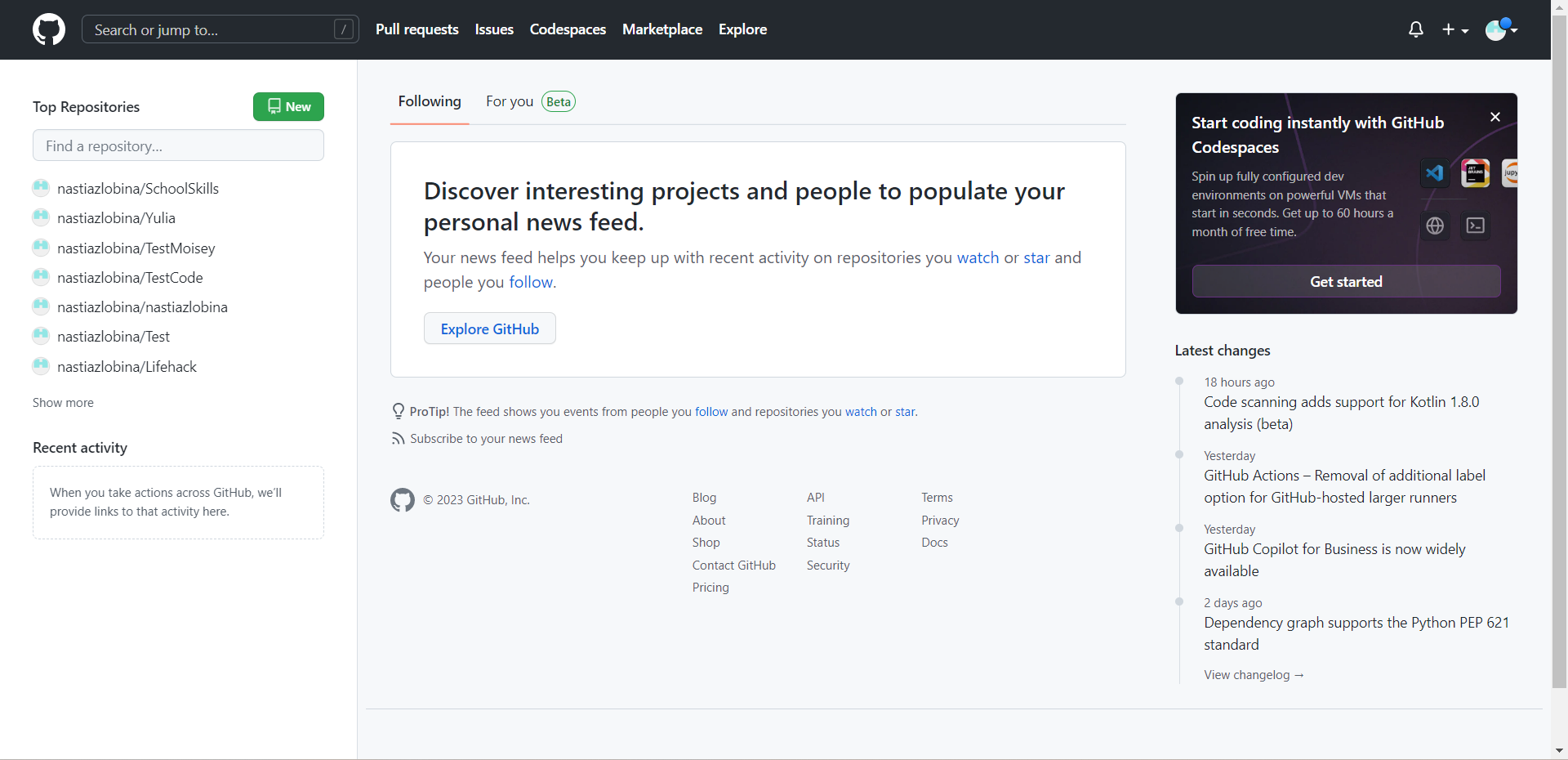
**GitHub**

Git — это консольная утилита, для отслеживания и ведения истории изменения файлов, в вашем проекте. Чаще всего его используют для кода, но можно и для других файлов. Например, для картинок - полезно для дизайнеров.

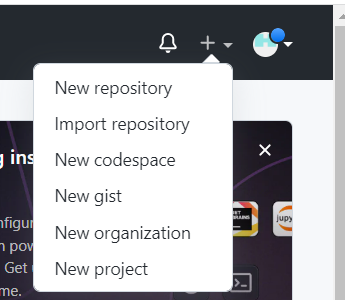
С помощью Git-a вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий.

Репозиторием называют хранилище вашего кода и историю его изменений. Git работает локально и все ваши репозитории хранятся в определенных папках на жестком диске.

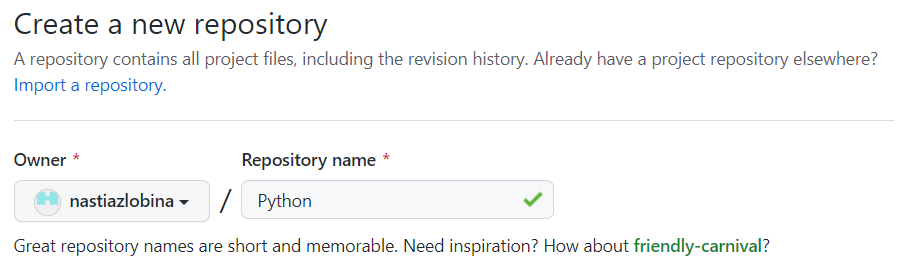
Для нашей дальнейшей работы, Вам необходимо зайти на сайт <https://github.com/> и зарегистрироваться. После регистрации, вам откроется личный кабинет.



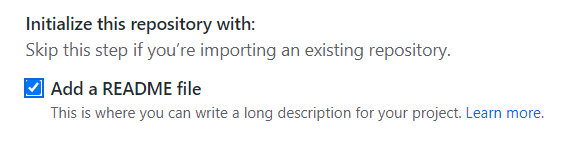
Для начала необходимо создать репозиторий, в котором будете сохранять все работы на протяжении всего курса.

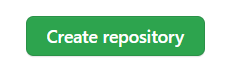


Назовите репозиторий как Python.

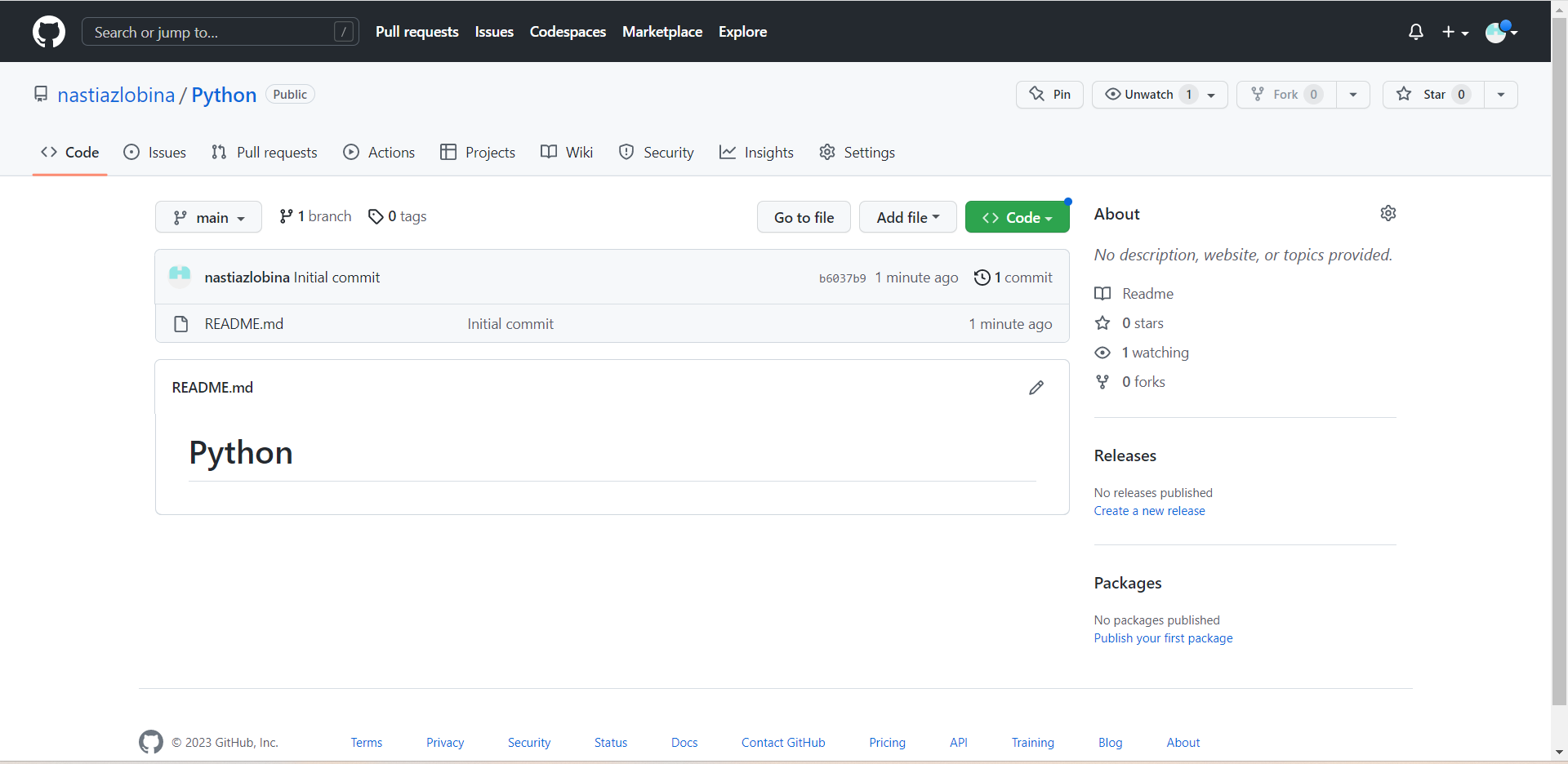


Нажмите флажок, а после создайте репозиторий.

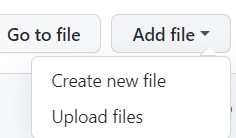




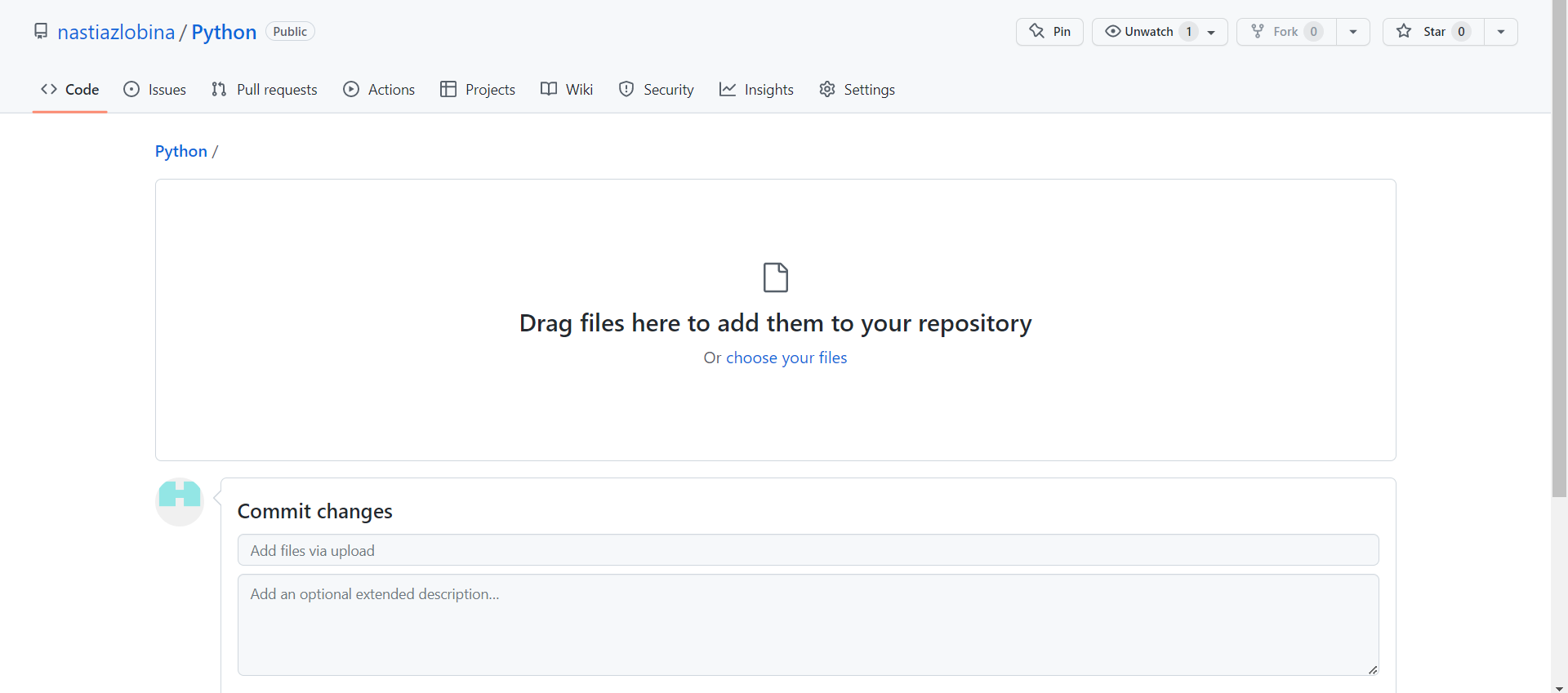
У Вас должно появиться окно с вашим репозиторием. Сюда Вы будете загружать свои работы.

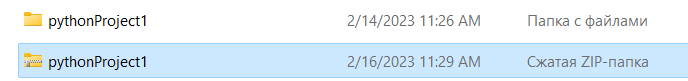


Для сохранения своей работы, необходимо нажать Add file – Upload file.



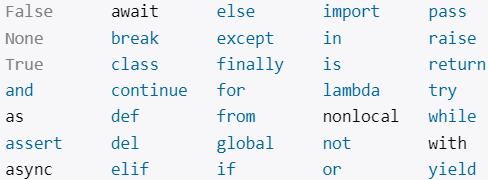
Выберите файл, который необходимо грузить. Если это программа, то она должна быть заархивирована. После того, как проект в архиве его можно загружать на Git.\



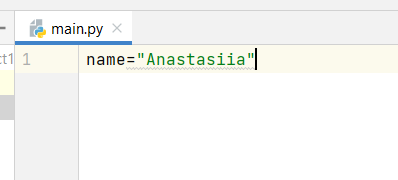


**Тема 2.1 Переменные. Ввод и вывод информации**

Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить:



Например, создадим переменную:



Здесь определена переменная **name**, которая хранит строку "**Anastasiia**". В Python применяется два типа наименования переменных: **camel case** и **underscore notation**.

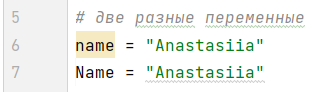
**Camel case** подразумевает, что каждое новое подслова в наименовании переменной начинается с большой буквы. Например:



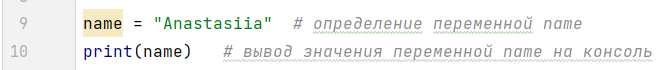
**Underscore notation** подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:



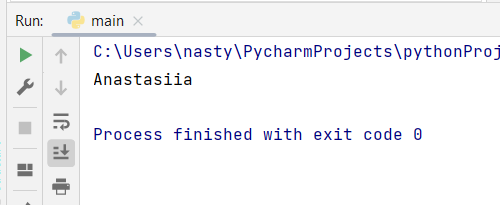
И также надо учитывать регистрозависимость, поэтому переменные **name** и **Name** будут представлять разные объекты.



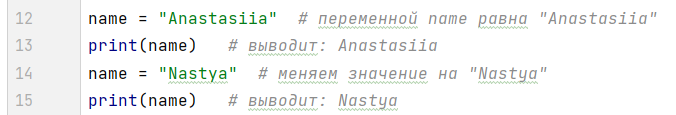
Определив переменную, мы можем использовать в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции **print**:



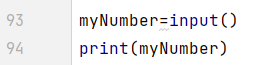
Результат работы программы:



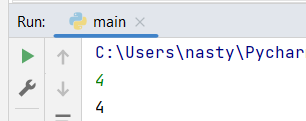
Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:



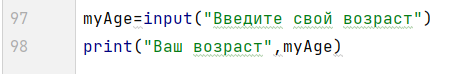
Очень часто при написании программы нужно запрашивать какую-либо информацию у пользователя. И после того, как он введет необходимые данные, их нужно сохранить в программе. Для обработки пользовательского ввода в python существует функция **input()**.

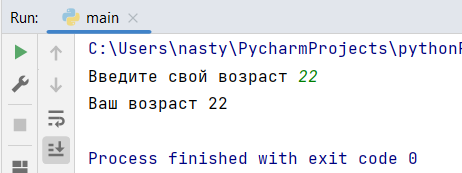


После запуска программы, если вы используете **input()**, то в консоль необходимо ввести какое-либо значение, чтобы оно записалось в переменную.



Также можно в **input()** вписывать текст, который будет отображаться для удобства работы с программой.





**Задание по теме 2.1 Переменные. Ввод и вывод информации**

Выберите тип наименования переменных, который будете использовать в дальнейшей работе. В созданном проекте из темы 1.1 создайте переменные и задайте им значения ваших имен или фамилий. Выведите на консоль созданные переменные.

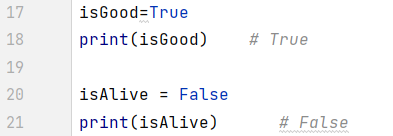
Введите значения в переменные с консоли и выведите их обратно на консоль.

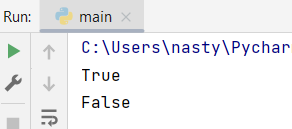
**Тема 2.2 Типы данных**

Переменная хранит данные одного из типов данных. В Python существует множество различных типов данных. В данном случае рассмотрим только самые базовые типы: **bool, int, float** и **str.**

**Логические значения**

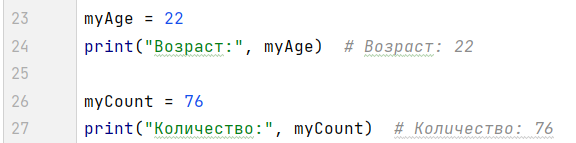
Тип **bool** представляет два логических значения: **True** (верно, истина) или **False** (неверно, ложь). Значение **True** служит для того, чтобы показать, что что-то истинно. Тогда как значение **False**, наоборот, показывает, что что-то ложно. Пример переменных данного типа:

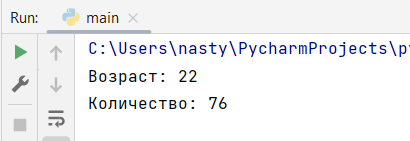




**Целые числа**

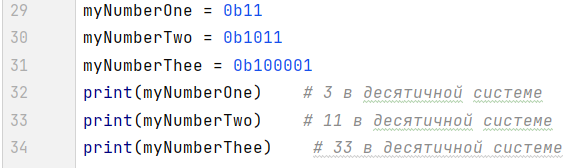
Тип int представляет целое число, например, 5, 9, 14, 55. Пример



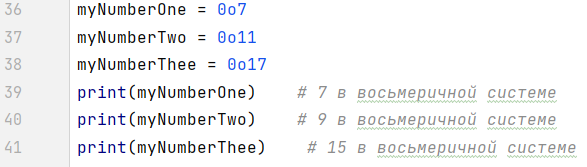


Для вывода переменных вместе с текстом, достаточно написать информацию в кавычках, поставить запятую и написать переменную, которую необходимо вывести.

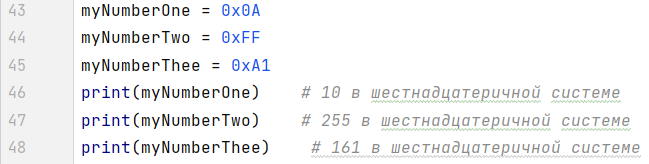
По умолчанию стандартные числа расцениваются как числа в десятичной системе. Но Python также поддерживает числа в двоичной, восьмеричной и шестнадцатеричной системах. Для указания, что число представляет двоичную систему, перед числом ставится префикс **0b**:



Для указания, что число представляет восьмеричную систему, перед числом ставится префикс **0o**:



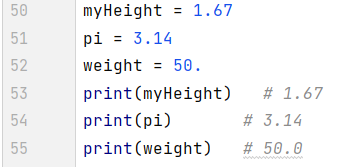
Для указания, что число представляет шестнадцатеричную систему, перед числом ставится префикс **0x**:

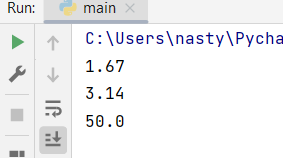


Стоит отметить, что в какой-бы системе мы не передали число в функцию **print** для вывода на консоль, оно по умолчанию будет выводиться в десятичной системе.

**Дробные числа**

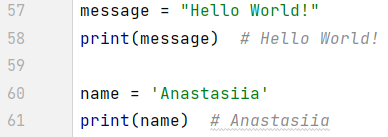
Тип float представляет число с плавающей точкой, например, 1.2 или 34.76. В качестве разделителя целой и дробной частей используется точка.

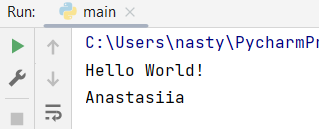




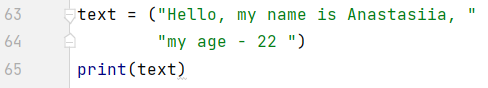
**Строки**

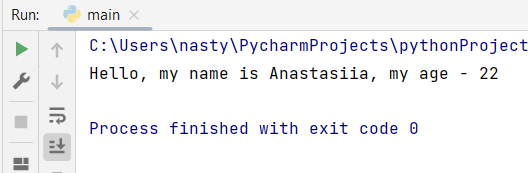
Тип **str** представляет строки. Строка представляет последовательность символов, заключенную в одинарные или двойные кавычки, например **"hello"** и **'hello'**.



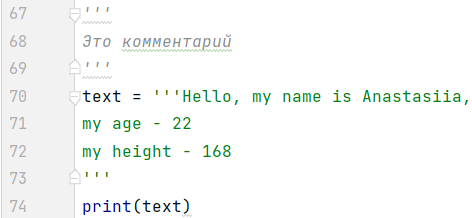


При этом если строка имеет много символов, ее можем разбить ее на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:





Если же мы хотим определить многострочный текст, то такой текст заключается в тройные двойные или одинарные кавычки:



При использовании тройных одинарных кавычек не стоит путать их с комментариями: если текст в тройных одинарных кавычках присваивается переменной, то это строка, а не комментарий.

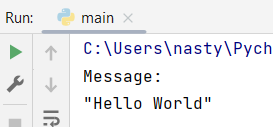
**Управляющие последовательности в строке**

Строка может содержать ряд специальных символов - управляющих последовательностей. Некоторые из них:

|  |  |
| --- | --- |
| **\** | позволяет добавить внутрь строки слеш |
| **\'** | позволяет добавить внутрь строки одинарную кавычку |
| **\"** | позволяет добавить внутрь строки двойную кавычку |
| **\n** | осуществляет переход на новую строку |
| **\t** | добавляет табуляцию (4 отступа) |

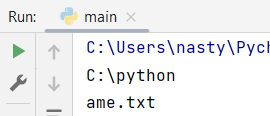
Применим несколько последовательностей:





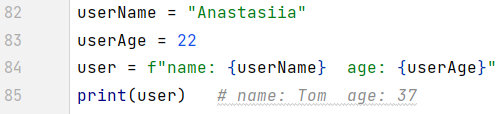
Хотя подобные последовательности могут нам помочь в некоторых делах, например, поместить в строку кавычку, сделать табуляцию, перенос на другую строку. Но они также могут и мешать. Например:



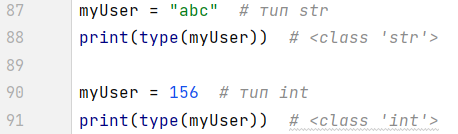


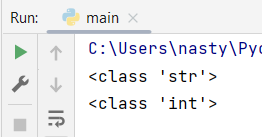
**Вставка значений в строку**

Python позволяет встраивать в строку значения других переменных. Для этого внутри строки переменные размещаются в фигурных скобках {}, а перед всей строкой ставится символ **f**:



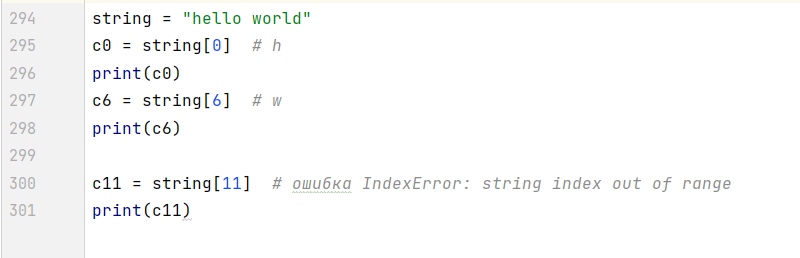
С помощью встроенной функции **type()** можно узнать текущий тип переменной:

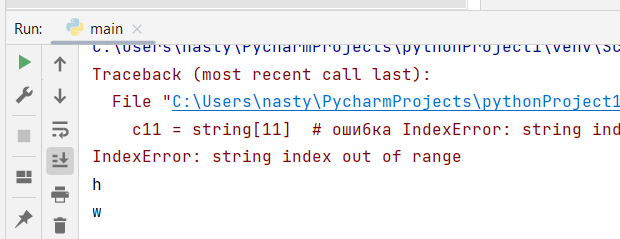




**Обращение к символам строки**

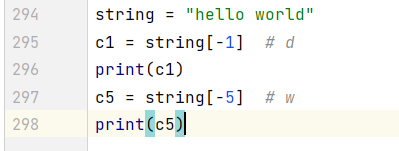
И мы можем обратиться к отдельным символам строки по индексу в квадратных скобках:

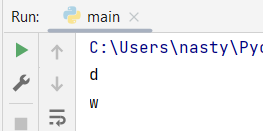
****

****

Индексация начинается с нуля, поэтому первый символ строки будет иметь индекс 0. А если мы попытаемся обратиться к индексу, которого нет в строке, то мы получим исключение IndexError. Например, в случае выше длина строки 11 символов, поэтому ее символы будут иметь индексы от 0 до 10.

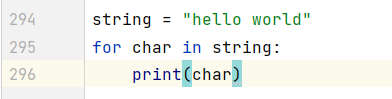
Чтобы получить доступ к символам, начиная с конца строки, можно использовать отрицательные индексы. Так, индекс -1 будет представлять последний символ, а -2 - предпоследний символ и так далее:

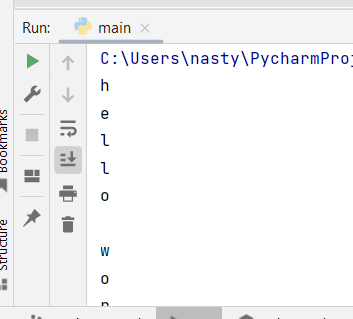




**Перебор строки**

С помощью цикла for можно перебрать все символы строки:





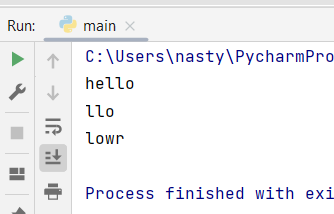
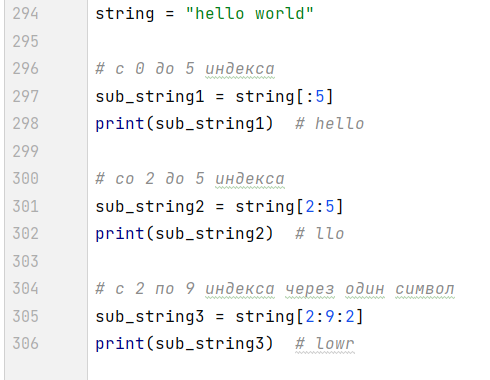
**Получение подстроки**

При необходимости мы можем получить из строки не только отдельные символы, но и подстроку. Для этого используется следующий синтаксис:

string[:end]: извлекается последовательность символов начиная с 0-го индекса по индекс end (не включая)

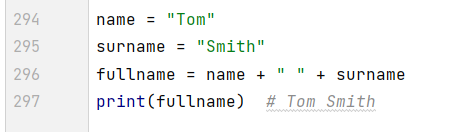
string[start:end]: извлекается последовательность символов начиная с индекса start по индекс end (не включая)

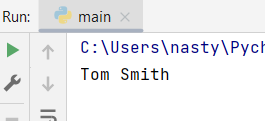
string[start:end:step]: извлекается последовательность символов начиная с индекса start по индекс end (не включая) через шаг step



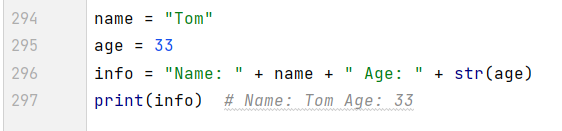
**Объединение строк**

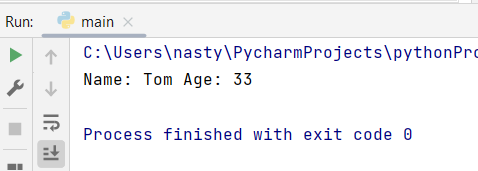
Одной из самых распространенных операций со строками является их объединение или конкатенация. Для объединения строк применяется операция сложения:





С объединением двух строк все просто, но что, если нам надо сложить строку и число? В этом случае необходимо привести число к строке с помощью функции str():





**Список**

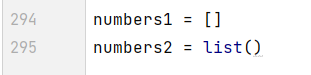
Для создания списка применяются квадратные скобки [], внутри которых через запятую перечисляются элементы списка. Например, определим список чисел:



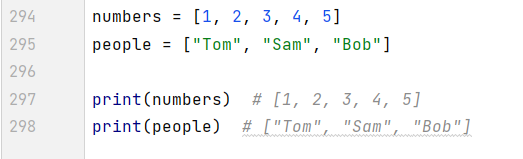
Подобным образом можно определять списки с данными других типов, например, определим список строк:

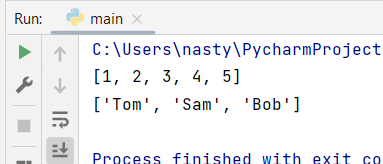


Также для создания списка можно использовать функцию-конструктор list():



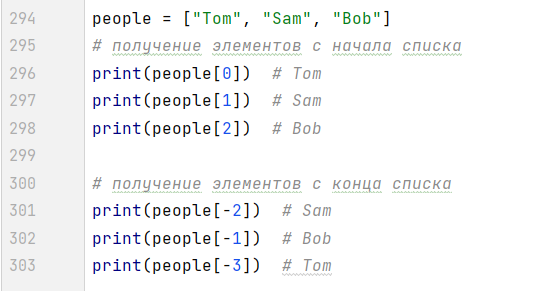
Для проверки элементов списка можно использовать стандартную функцию print, которая выводит содержимое списка в удобочитаемом виде:

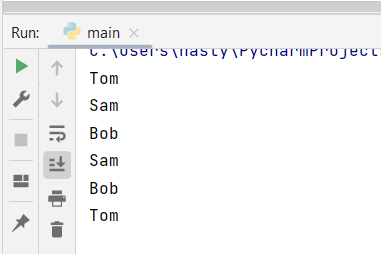




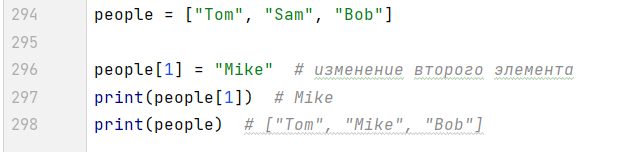
**Обращение к элементам списка**

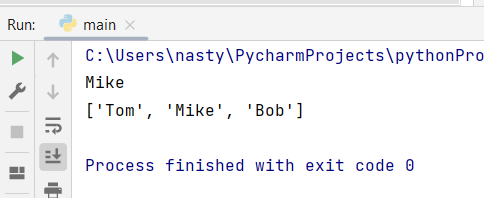
Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.





Для изменения элемента списка достаточно присвоить ему новое значение:

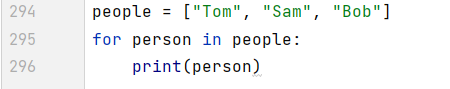


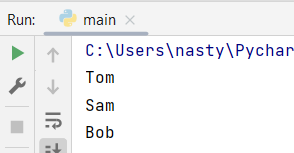


**Перебор элементов**

Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла for:



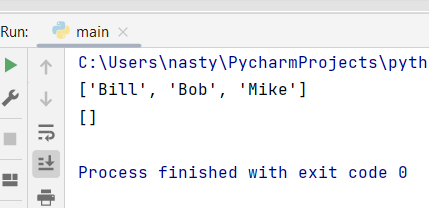


**Добавление и удаление элементов**

Для добавления элемента применяются методы append(), extend и insert, а для удаления - методы remove(), pop() и clear().

Использование методов:





**Методы и функции по работе со списками**

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

**append(item)**: добавляет элемент item в конец списка

**insert(index, item)**: добавляет элемент item в список по индексу index

**extend(items)**: добавляет набор элементов items в конец списка

**remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError

**clear()**: удаление всех элементов из списка

**index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError.

**pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.

**count(item)**: возвращает количество вхождений элемента item в список

**sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.

**reverse()**: расставляет все элементы в списке в обратном порядке

**copy()**: копирует список

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

**len(list)**: возвращает длину списка

**sorted(list, [key])**: возвращает отсортированный список

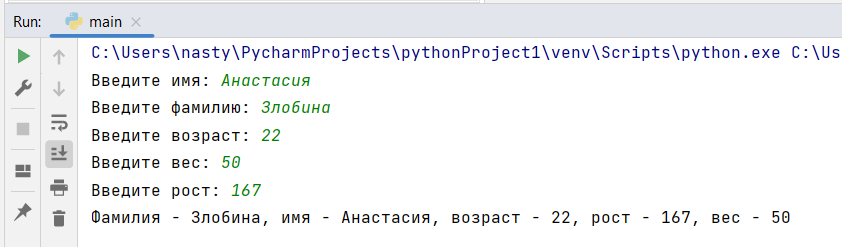
**min(list)**: возвращает наименьший элемент списка

**max(list)**: возвращает наибольший элемент списка

**contains(item)**: проверяет входит ли элемент в список.

**Задание по теме 2.2 Типы данных**

Введите с помощью консоли ваше имя, фамилию, возраст, рост и вес (данные могут не совпадать с реальностью). Не обязательно использовать именно эти параметры, это может быть все, что угодно (домашнее животное, друг, любимая игра и т.д.), включите свое воображение. Выведите одно предложение со всеми параметрами. Пример:



**Тема 3.1 Логические операции, операции сравнения**

Простейшие условные выражения представляют операции сравнения, которые сравнивают два значения. Python поддерживает следующие операции сравнения:

== Возвращает True, если оба операнда равны. Иначе возвращает False.

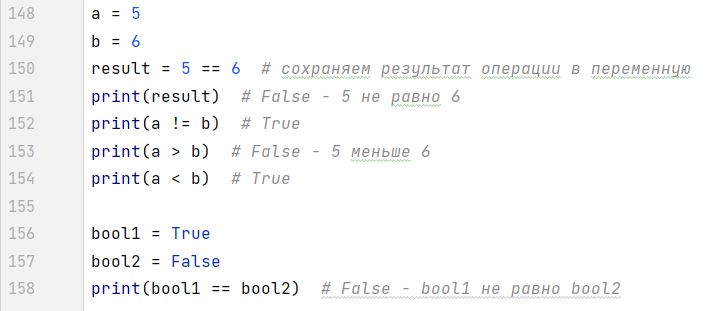
! = Возвращает True, если оба операнда НЕ равны. Иначе возвращает False.

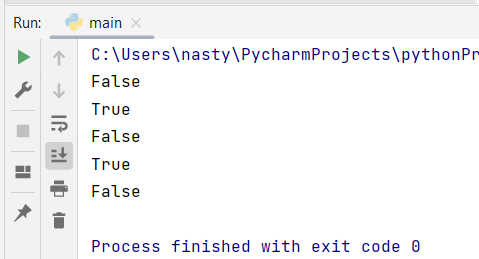
> (больше чем) Возвращает True, если первый операнд больше второго.

<(меньше чем) Возвращает True, если первый операнд меньше второго.

> = (больше или равно) Возвращает True, если первый операнд больше или равен второму.

<= (меньше или равно) Возвращает True, если первый операнд меньше или равен второму.

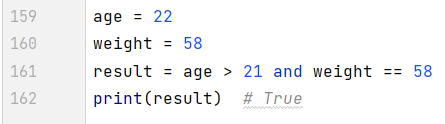
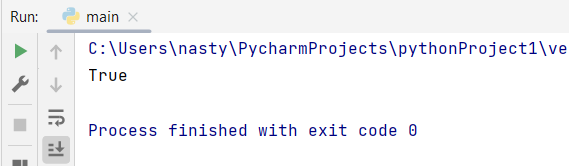




Для создания составных условных выражений применяются логические операции. В Python имеются следующие логические операторы:

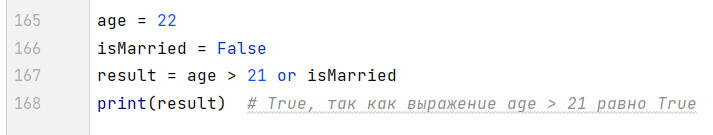
**Оператор and (логическое умножение) применяется к двум операндам:**

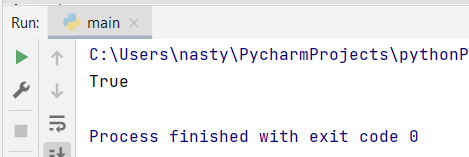
Сначала оператор **and** оценивает выражение x, и если оно равно **False**, то возвращается его значение. Если оно равно **True**, то оценивается второй операнд - y и возвращается значение y.

**or (логическое сложение) также применяется к двум операндам:**

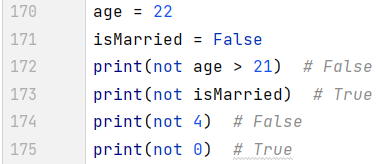
Сначала оператор or оценивает выражение x, и если оно равно True, то возвращается его значение. Если оно равно False, то оценивается второй операнд - y и возвращается значение y. Например

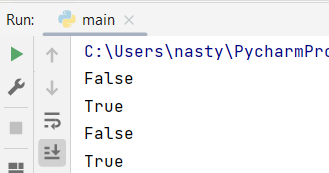




**not (логическое отрицание)**

Возвращает True, если выражение равно False



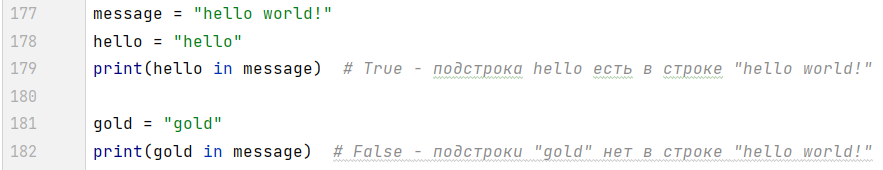
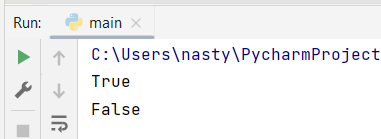


**Оператор in**

Оператор in возвращает True если в некотором наборе значений есть определенное значение. Он имеет следующую форму:

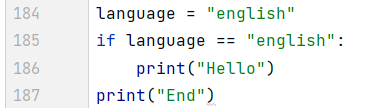
значение in набор\_значений

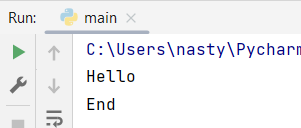
Например, строка представляет набор символов. И с помощью оператора in мы можем проверить, есть ли в ней какая-нибудь подстрока:

**Условная конструкция if**

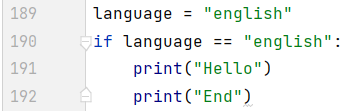
В самом простом виде после ключевого слова if идет логическое выражение. И если это логическое выражение возвращает True, то выполняется последующий блок инструкций, каждая из которых должна начинаться с новой строки и должна иметь отступы от начала выражения if (отступ желательно делать в 4 пробела или то количество пробелов, которое кратно 4):

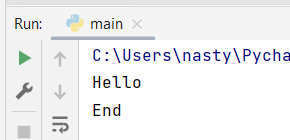




Обратите внимание в коде на последнюю строку, которая выводит сообщение "End". Она не имеет отступов от начала строки, поэтому она не принадлежит к блоку if и будет выполняться в любом случае, даже если выражение в конструкции if возвратит False.

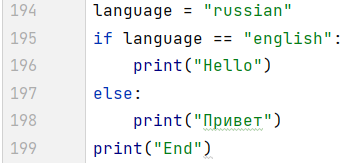
Но если бы мы поставили бы отступы, то она также принадлежала бы к конструкции if:

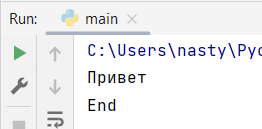




**Блок else**

Если вдруг нам надо определить альтернативное решение на тот случай, если выражение в if возвратит False, то мы можем использовать блок else:





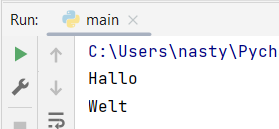
Если выражение language == "english" возвращает True, то выполняется блок if, иначе выполняется блок else. И поскольку в данном случае условие language == "english" возвращает False, то будут выполняться инструкция из блока else.

Причем инструкции блока else также должны имет отступы от начала строки. Например, в примере выше print("End") не имеет отступа, поэтому она не входит в блок else и будет выполнятьься вне зависимости, чему равно условие language == "english".

**Elif**

Если необходимо ввести несколько альтернативных условий, то можно использовать дополнительные блоки elif, после которого идет блок инструкций.



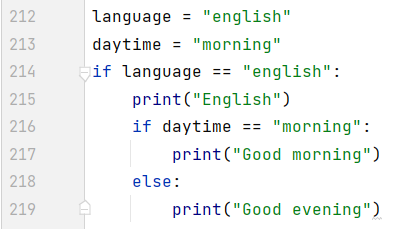


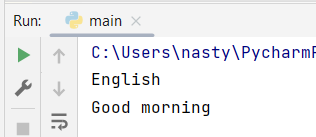
Сначала Python проверяет выражение if. Если оно равно True, то выполнениются инструкции из блока if. Если это условие возвращает False, то Python проверяет выражение из elif.

Если выражение после elif равно True, то выполняются инструкции из блока elif. Но если оно равно False то выполняются инструкции из блока else

**Вложенные конструкции if**

Конструкция if в свою очередь сама может иметь вложенные конструкции if:





Здесь конструкция if содержит вложенную конструкцию if/else. То есть если переменная language равна "english", тогда вложенная конструкция if/else дополнительно проверяет значение переменной daytime - равна ли она строке "morning" ли нет.

**Тема 3.2 Цикл while. Выход из цикла. break и continue. Вложенные циклы**

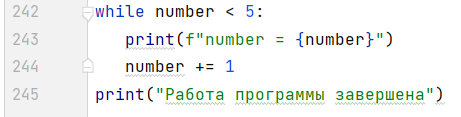
Циклы позволяют выполнять некоторое действие в зависимости от соблюдения некоторого условия.

Цикл while проверяет истинность некоторого условия, и если условие истинно, то выполняет инструкции цикла. Он имеет следующее формальное определение:



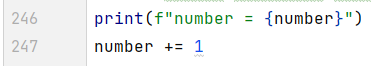
После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала ключевого слова while.



В данном случае цикл while будет выполняться, пока переменная number меньше 5.

Сам блок цикла состоит из двух инструкций:



Обратите внимание, что они имеют отступы от начала оператора while - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. .

Также обратите внимание, что последняя инструкция print("Работа программы завершена") не имеет отступов от начала строки, поэтому она не входит в цикл while.

Весь процесс цикла можно представить следующим образом:

Сначала проверяется значение переменной number - меньше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное выполнение блока инструкций цикла называется итерацией. То есть таким образом, в цикле выполняется первая итерация.

Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация.

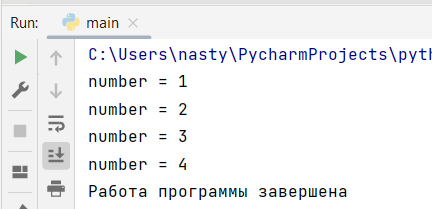
Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4. То есть выполняется третья итерация.

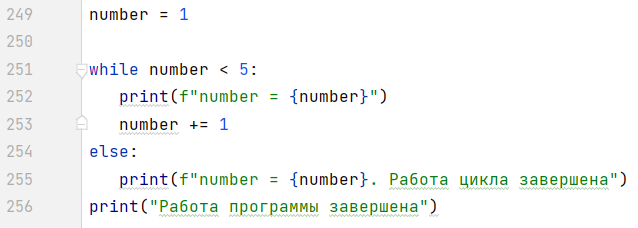
Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла

Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация.

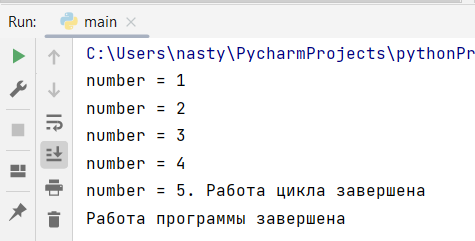
И вновь проверяется условие number < 5. Но теперь оно равно False, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации



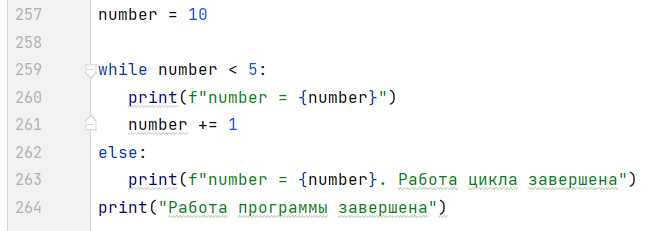
Для цикла while также можно определить дополнительный блок else, инструкции которого выполняются, когда условие равно False:



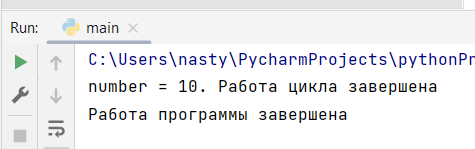
То есть в данном случае сначала проверяется условие и выполняются инструкции while. Затем, когда условие становится равным False, выполняются инструкции из блока else. Обратите внимание, что инструкции из блока else также имеют отступы от начала конструкции цикла. В итоге в данном случае мы получим следующий консольный вывод:



Блок else может быть полезен, если условие изначально равно False, и мы можем выполнить некоторые действия по этому поводу:

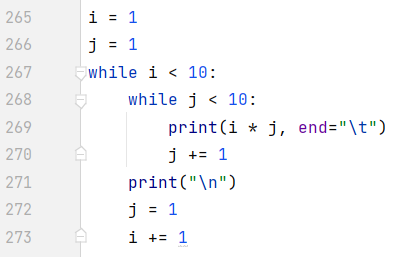


В данном случае условие number < 5 изначально равно False, поэтому цикл не выполняет ни одной итерации и сразу переходит в блоку else.



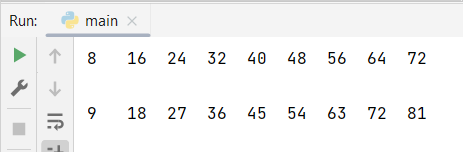
**Вложенные циклы**

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения:



Внешний цикл while i < 10: срабатывает 9 раз пока переменная i не станет равна 10. Внутри этого цикла срабатывает внутренний цикл while j < 10:. Внутренний цикл также срабатывает 9 раз пока переменная j не станет равна 10. Причем все 9 итераций внутреннего цикла срабатывают в рамках одной итерации внешнего цикла.

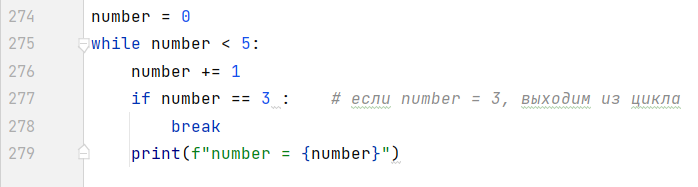
В каждой итерации внутреннего цикла на консоль будет выводится произведение чисел i и j. Затем значение переменной j увеличивается на единицу. Когда внутренний цикл закончил работу, значений переменной j сбрасывается в 1, а значение переменной i увеличивается на единицу и происходит переход к следующей итерации внешнего цикла. И все повторяется, пока переменная i не станет равна 10. Соответственно внутренний цикл сработает всего 81 раз для всех итераций внешнего цикла. В итоге мы получим следующий консольный вывод:



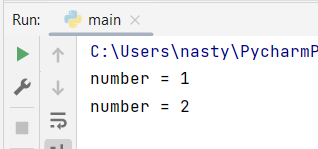
**Выход из цикла. break и continue**

Для управления циклом мы можем использовать специальные операторы break и continue. Оператор break осуществляет выход из цикла. А оператор continue выполняет переход к следующей итерации цикла.

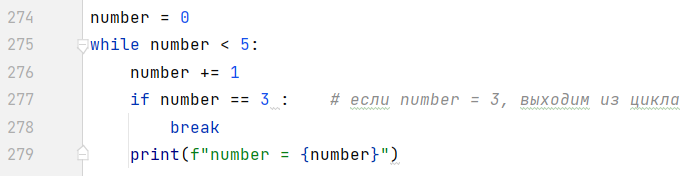
Оператор break может использоваться, если в цикле образуются условия, которые несовместимы с его дальнейшим выполнением. Рассмотрим следующий пример:



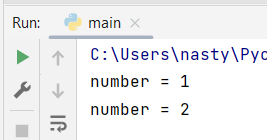
Здесь цикл while проверяет условие number < 5. И пока number не равно 5, предполагается, что значение number будет выводиться на консоль. Однако внутри цикла также проверяется другое условие: if number == 3. То есть, если значение number равно 3, то с помощью оператора break выходим из цикла. И в итоге мы получим следующий консольный вывод:



В отличие от оператора break оператор continue выполняет переход к следующей итерации цикла без его завершения. Например, в предыдущем примере заменим break на continue:



И в этом случае если значение переменной number равно 3, последующие инструкции после оператора continue не будут выполняться:



**Тема 3.5 Цикл for.**

Другой тип циклов представляет конструкция for. Этот цикл пробегается по набору значений, помещает каждое значение в переменную, и затем в цикле мы можем с этой переменной производить различные действия. Формальное определение цикла for:

**for** переменная **in** набор\_значений:

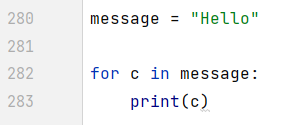
инструкции.

После ключевого слова for идет название переменной, в которую будут помещаться значения. Затем после оператора in указывается набор значений и двоеточие.

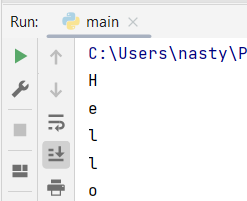
А со следующей строки располагается блок инструкций цикла, которые также должны иметь отступы от начала цикла.

При выполнении цикла Python последовательно получает все значения из набора и передает их переменную. Когда все значения из набора будут перебраны, цикл завершает свою работу.

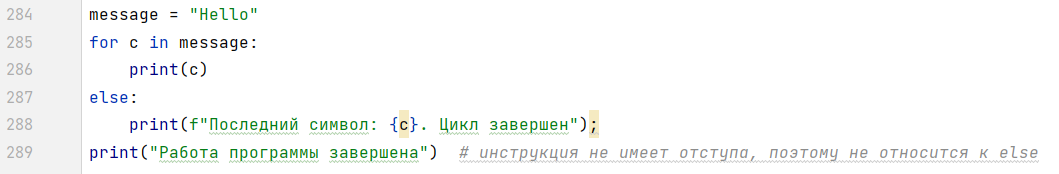
В качестве набора значений, например, можно рассматривать строку, которая по сути представляет набор символов. Посмотрим на примере:



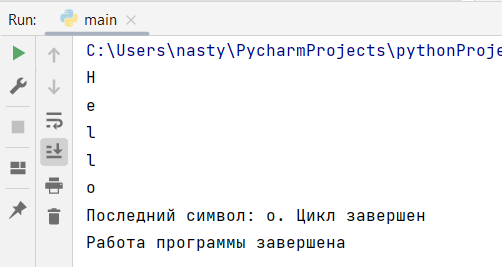
В цикле определяется переменную c, после оператора in в качестве перебираемого набора указана переменная message, которая хранит строку "Hello". В итоге цикл for будет перебираеть последовательно все символы из строки message и помещать их в переменную c. Блок самого цикла состоит из одной инструкции, которая выводит значение переменной с на консоль. Консольный вывод программы:



Цикл for также может иметь дополнительный блок else, который выполняется после завершения цикла:



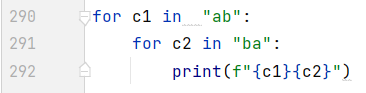
В данном случае мы получим следующий консольный вывод:



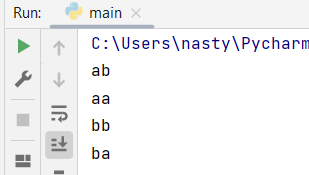
Стоит отметить, что блок else имеет доступ ко всем переменным, которые определены в цикле for.

**Вложенные циклы**

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения:



В данном случае внешний цикл проходит по строке "ab" и каждый символ помещает в переменную c1. Внутренний цикл проходит по строке "ba", помещает каждый символ строки в переменную c2 и выводит сочетание обоих символов на консоль. То есть в итоге мы получим все возможные сочетания символов a и b:

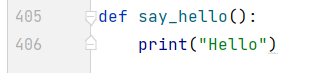


**Тема 3.6 Функции.**

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы. В предыдущих лекциях уже использовались функции. В частности, функция print(), которая выводит некоторое значение на консоль. Python имеет множество встроенных функций и позволяет определять свои функции.

Определение функции начинается с выражения def, которое состоит из имени функции, набора скобок с параметрами и двоеточия. Параметры в скобках необязательны. А со следующей строки идет блок инструкций, которые выполняет функция. Все инструкции функции имеют отступы от начала строки.

Например, определение простейшей функции:



Функция называется say\_hello. Она не имеет параметров и содержит одну единственную инструкцию, которая выводит на консоль строку "Hello".

Для вызова функции указывается имя функции, после которого в скобках идет передача значений для всех ее параметров.



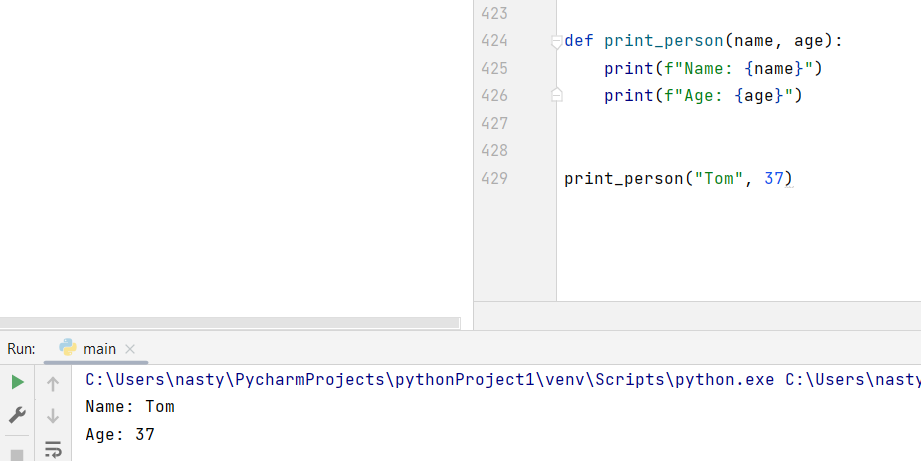
**Параметры функции**

Функция может принимать параметры. Через параметры в функцию можно передавать данные. Банальный пример - функция print(), которая с помощью параметра принимает значение, выводимое на консоль.

Теперь определим и используем свою функцию с параметрами:



При вызове функции значения передаются параметрам по позиции. Например, определим и вызовем функцию с несколькими параметрами:



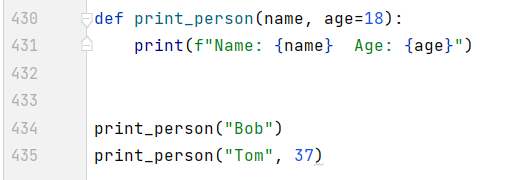
Здесь функция print\_person принимает два параметра: name и age. При вызове функции. Первое значение - "Tom" передается первому параметру, то есть параметру name. Второе значение - 37 передается второму параметру - age. И внутри функции значения параметров выводятся на консоль.

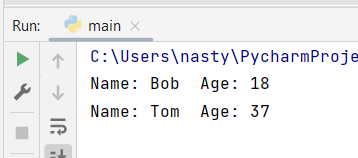
**Значения по умолчанию**

Некоторые параметры функции мы можем сделать необязательными, указав для них значения по умолчанию при определении функции. Например:



Здесь параметр name является необязательным. И если мы не передаем при вызове функции для него значение, то применяется значение по умолчанию, то есть строка "Tom". Если функция имеет несколько параметров, то необязательные параметры должны идти после обязательных. Например

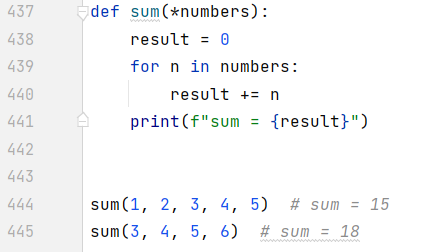


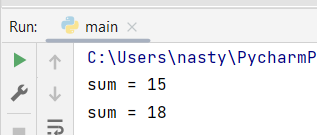


Здесь параметр age является необязательным и по умолчанию имеет значение 18. Перед ним расположен обязательный параметр name. Поэтому при вызове функции мы можем не передавать значение параметру age, но параметру name передать значение необходимо.

**Неопределенное количество параметров**

С помощью символа звездочки можно определить параметр, через который можно передавать неопределенное количество значений. Это может быть полезно, когда мы хотим, чтобы функция получала несколько значений, но мы точно не знаем, сколько именно. Например, определим функцию подсчета суммы чисел:





В данном случае функция sum принимает один параметр - \*numbers, но звездочка перед названием параметра указывает, что фактически на место этого параметра мы можем передать неопределенное количество значений или набор значений. В самой функции с помощью цикла for можно пройтись по этому набору, получить каждое значение из этого набора в переменную n и произвести с ним какие-нибудь действия. Например, в данном случае вычисляется сумма переданных чисел.

**Тема 3.7. Сторонние библиотеки**

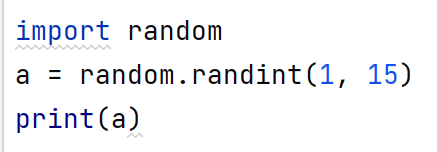
Язык программирования Python имеет много встроенных функций. Однако их не хватает для решения всех видов задач, поэтому программисты добавляют инструменты, подключая модули.

**Обычное подключение — import**

Подключать модули желательно в самом вверху скрипта, используя ключевое слово «import», например import random.

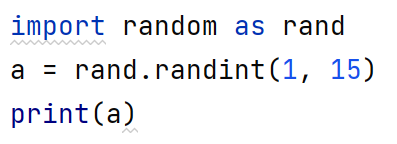
После подключения программа получает доступ ко всем функциям, методам и классам, содержащимся в нём.

Этот способ не допускает пересечения имён, то есть программист может использовать одно и то же имя функции в скрипте, точно такое же, как и в подключаемой библиотеке и не бояться, что после её подключения, функция будет переопределена.



**Использование псевдонимов — as**

Некоторые модули имеют длинное и неудобное название. Для удобства и сокращения количества кода программист может заменить его на своё.



**Импорт компонентов — from**

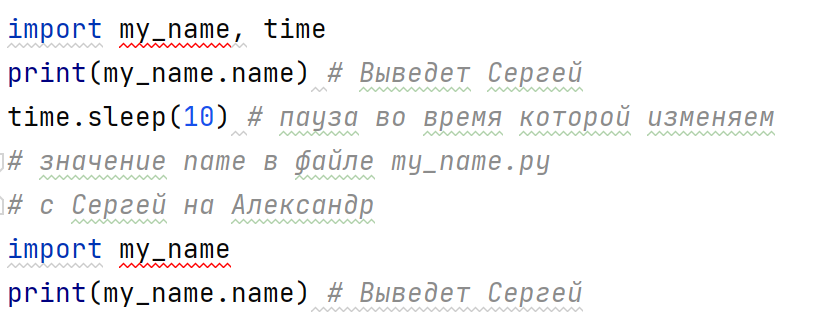
Чтобы не захламлять программу большим количеством неиспользуемых инструментов, можно подключать не весь модуль, а какую-то его часть.

Например, программист хочет использовать только одну функцию из математической библиотеки math. Если он подключит всю библиотеку, то в скрипт добавится более 40 функций, которые будут занимать место. Чтобы добавить в проект какую-то часть, используют ключевое слово from:

**Перезагрузка библиотеки**

Если программист после импорта, изменит в файле, который импортировал что-либо, а потом снова его импортирует, основная программа не будет видеть этих изменений.

Всё потому, что при импорте библиотека кешируется, когда её пытаются импортировать снова, интерпретатор Python просто использует сохранённую в кэше копию.



**Библиотеки для создания бота**

Для создания Telegram-ботов на Python существует несколько десятков библиотек. Они различаются популярностью, размером комьюнити и функциональностью. Рассмотрим самые популярные.

**aiogram**

Современная библиотека, набирающая популярность. Работает с асинхронным подходом к выполнению кода. Это позволяет не останавливать работу бота в ожидании ответа пользователя. У aiogram подробная документация и большое русскоязычное комьюнити.

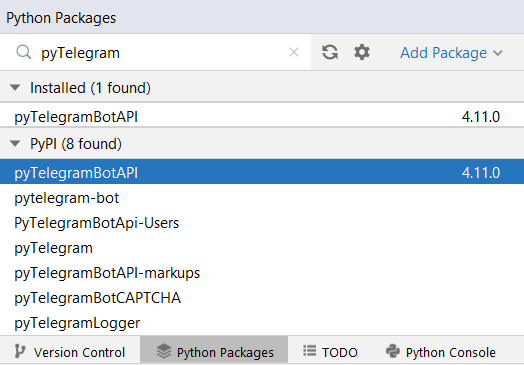
**python-telegram-bot**

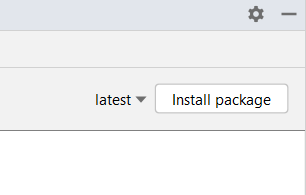
Одна из первых библиотек для создания ботов. Отличается от aiogram синхронным подходом к работе, то есть при ожидании ответа от пользователя выполнение кода останавливается.

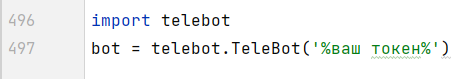
**TeleBot**

Библиотека для создания простых ботов, позволяющая работать с асинхронным и синхронным подходом на выбор. Подходит для небольших проектов.

**Подключение TeleBot**



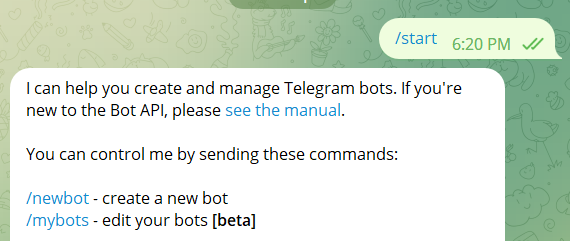




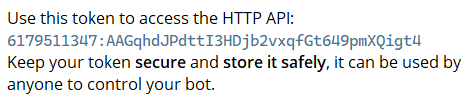
**Тема 4. Телеграм бот**

В телеграме ищем BotFather.





Пишем /start и выбираем команду /newbot, после отправляем название вашего бота и получаем токен .



После приступаем к созданию эхо бота.

import telebot

# Создаем экземпляр бота

bot = telebot.TeleBot('Здесь впиши токен, полученный от @botfather')

# Функция, обрабатывающая команду /start

@bot.message\_handler(commands=["start"])

def start(m, res=False):

bot.send\_message(m.chat.id, 'Я на связи. Напиши мне что-нибудь )')

# Получение сообщений от юзера

@bot.message\_handler(content\_types=["text"])

def handle\_text(message):

bot.send\_message(message.chat.id, 'Вы написали: ' + message.text)

# Запускаем бота

bot.polling(none\_stop=True, interval=0)

Теперь, когда мы создали бота с минимальным функционалом, применим работу с листами к нашему боту. Пример сделан по листу с цветами, вы используете свои листы

**ПРИМЕР С ЛИСТАМИ БЕЗ ИСПОЛЬЗОВАНИЯ БОТА**

colors=['Красный','Синий','Белый','Оранжевый','Зелёный','Жёлтый']  
colorsreturn=[]  
for item in colors:  
 colorsreturn.append(item)  
*# Строка для записи рандомного цвета*str=''  
*#Строка для записи цветов от пользователя*item=''

def RandomElement():  
 if len(colorsreturn)!=0:  
 index = random.randint(0, len(colorsreturn) - 1)  
 str = colorsreturn[index]  
 colorsreturn.remove(colorsreturn[index])  
 return str  
 else:  
 for item in colors:  
 colorsreturn.append(item)  
 index = random.randint(0, len(colorsreturn) - 1)  
 str = colorsreturn[index]  
 colorsreturn.remove(colorsreturn[index])  
 return str  
*# Метод для добавления цвета с проверкой на существующие цвета*def AddElement(item=''):  
 if item=='':  
 item=input('Введите цвет для добавления: ')  
 if not colors.\_\_contains\_\_(item):  
 colors.append(item)  
 colorsreturn.append(item)  
 print('Цвет добавлен')  
 else:  
 print('Цвет уже есть')  
 AddElement()  
*# Метод для удаления цвета с проверкой на существующие цвета*def RemoveElement():  
 item=input('Введите цвет для удаления: ')  
 if colors.\_\_contains\_\_(item):  
 colors.remove(item)  
 colorsreturn.remove(item)  
 print('Цвет удалён')  
 else:  
 print('Цвета не существует в списке, хотите добавлю?(Скажите да)')  
 if input()=='да':  
 AddElement(item)  
 RemoveElement()  
  
*# Бесконечный цикл для рандомных значений*while True:  
 print('Нажмите 1 чтобы зарандомить :\nНажмите 2 чтобы добавить элемент:\nНажмите 3 чтобы удалить элемент:')  
 number=input()  
 if number == '1':  
 print(RandomElement())  
 if number == '2':  
 AddElement()  
 if number == '3':  
 RemoveElement()

**ПРИМЕР С ЛИСТАМИ С ИСПОЛЬЗОВАНИЕМ БОТА**

import telebot  
*# Создаем экземпляр бота*bot = telebot.TeleBot('ВАШ ТОКЕН')  
*# Список цветов и список, из которого будут генерироваться цвета*colors=['Красный','Синий','Белый','Оранжевый','Зелёный','Жёлтый']  
colorsreturn=[]  
for item in colors:  
 colorsreturn.append(item)  
*# Строка для записи рандомного цвета*str=''  
*#Строка для записи цветов от пользователя*item=''

*# Функция, обрабатывающая команду /start*@bot.message\_handler(commands=["start"])  
def start(m, res=False):  
 bot.send\_message(m.chat.id, 'Нажмите 1 чтобы зарандомить:\nНажмите 2 чтобы добавить элемент:\nНажмите 3 '  
 'чтобы удалить элемент:')

*# Получение сообщений от юзера*@bot.message\_handler(content\_types=["text"])  
def handle\_text(message):  
 if message.text == '1':  
 bot.send\_message(message.chat.id, 'Я зарандомил: ' + RandomElement())  
 if message.text == '2':  
 bot.send\_message(message.chat.id, 'Напишите цвет ')  
 bot.register\_next\_step\_handler(message,AddElement)  
 if message.text == '3':  
 bot.send\_message(message.chat.id, 'Напишите цвет, который надо удалить')  
 bot.register\_next\_step\_handler(message, RemoveElement)  
 *# Метод для рандомного цвета*def RandomElement():  
 if len(colorsreturn)!=0:  
 index = random.randint(0, len(colorsreturn) - 1)  
 str = colorsreturn[index]  
 colorsreturn.remove(colorsreturn[index])  
 return str  
 else:  
 for item in colors:  
 colorsreturn.append(item)  
 index = random.randint(0, len(colorsreturn) - 1)  
 str = colorsreturn[index]  
 colorsreturn.remove(colorsreturn[index])  
 return str  
 *# Метод для добавление цвета с проверкой на существующие цвета*def AddElement(message):  
 global item  
 item=message.text  
 if not colors.\_\_contains\_\_(item):  
 colors.append(item)  
 colorsreturn.append(item)  
 bot.send\_message(message.from\_user.id, f'Добавил цвет {message.text}');  
 else:  
 bot.send\_message(message.chat.id, f'Цвет {message.text} существует, напишите новый')  
 bot.register\_next\_step\_handler(message,AddElement)  
*# Метод для удаления цвета с проверкой на существующие цвета*def RemoveElement(message):  
 global item  
 item = message.text  
 if colors.\_\_contains\_\_(item):  
 colors.remove(item)  
 colorsreturn.remove(item)  
 bot.send\_message(message.from\_user.id, f'Удалил цвет {message.text}');  
 else:  
 bot.send\_message(message.from\_user.id, f'Цвета не существует в списке, хотите добавлю?(Скажите да)');  
 bot.register\_next\_step\_handler(message, AddElement)  
*# Запускаем бота*bot.polling(none\_stop=True, interval=0)