**Методический материал по курсу**

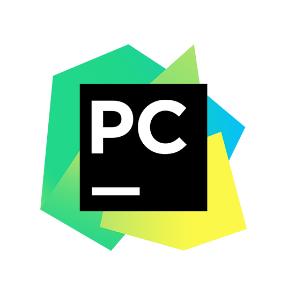
«16199 Оператор электронно-вычислительных и вычислительных машин с основами языка программирования Python»

**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;

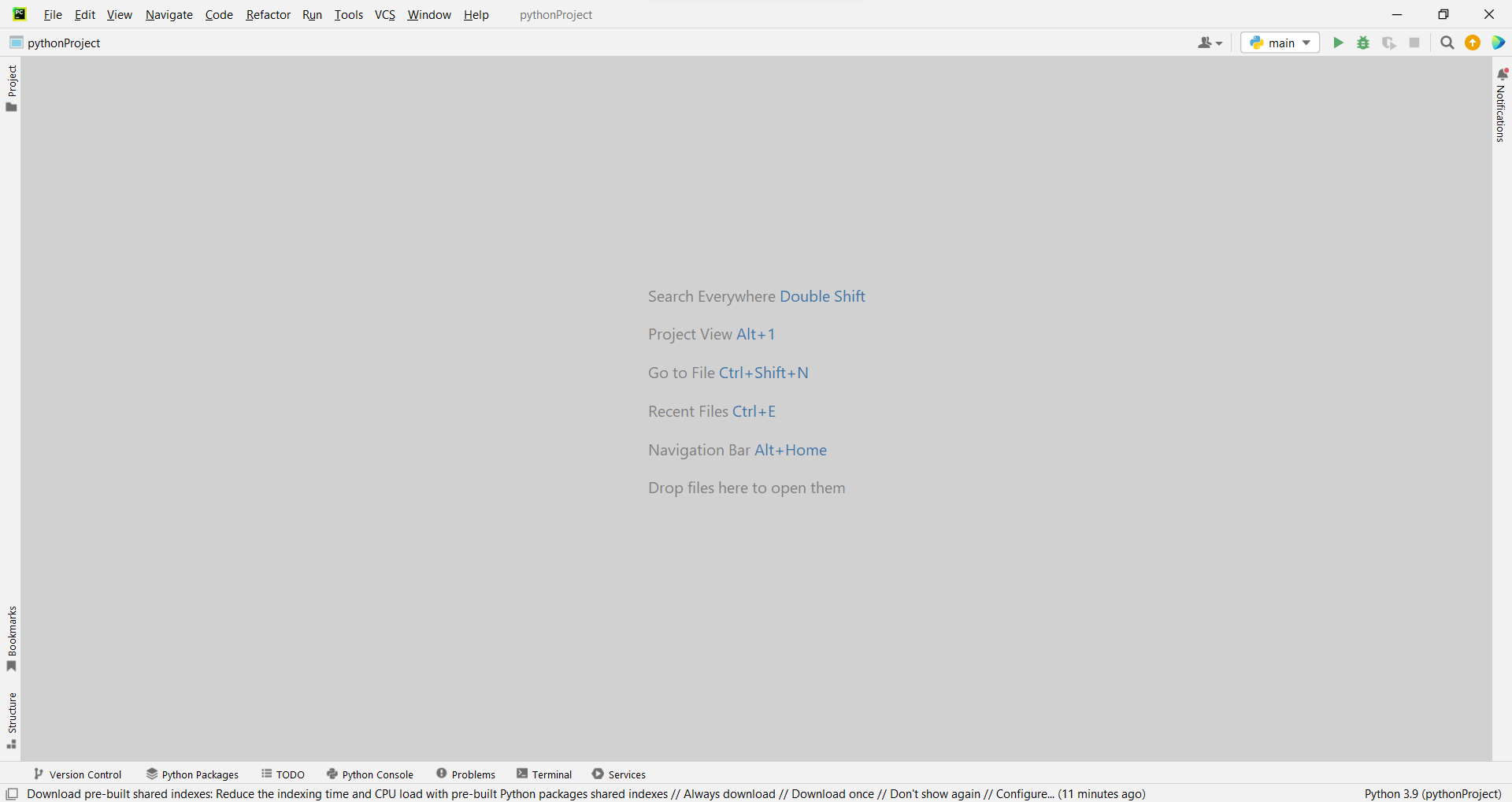
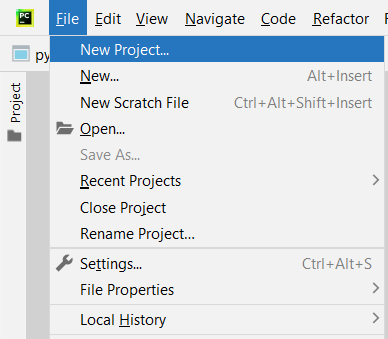


Рисунок 2

1. Нажать в верхней панели File – New Project;

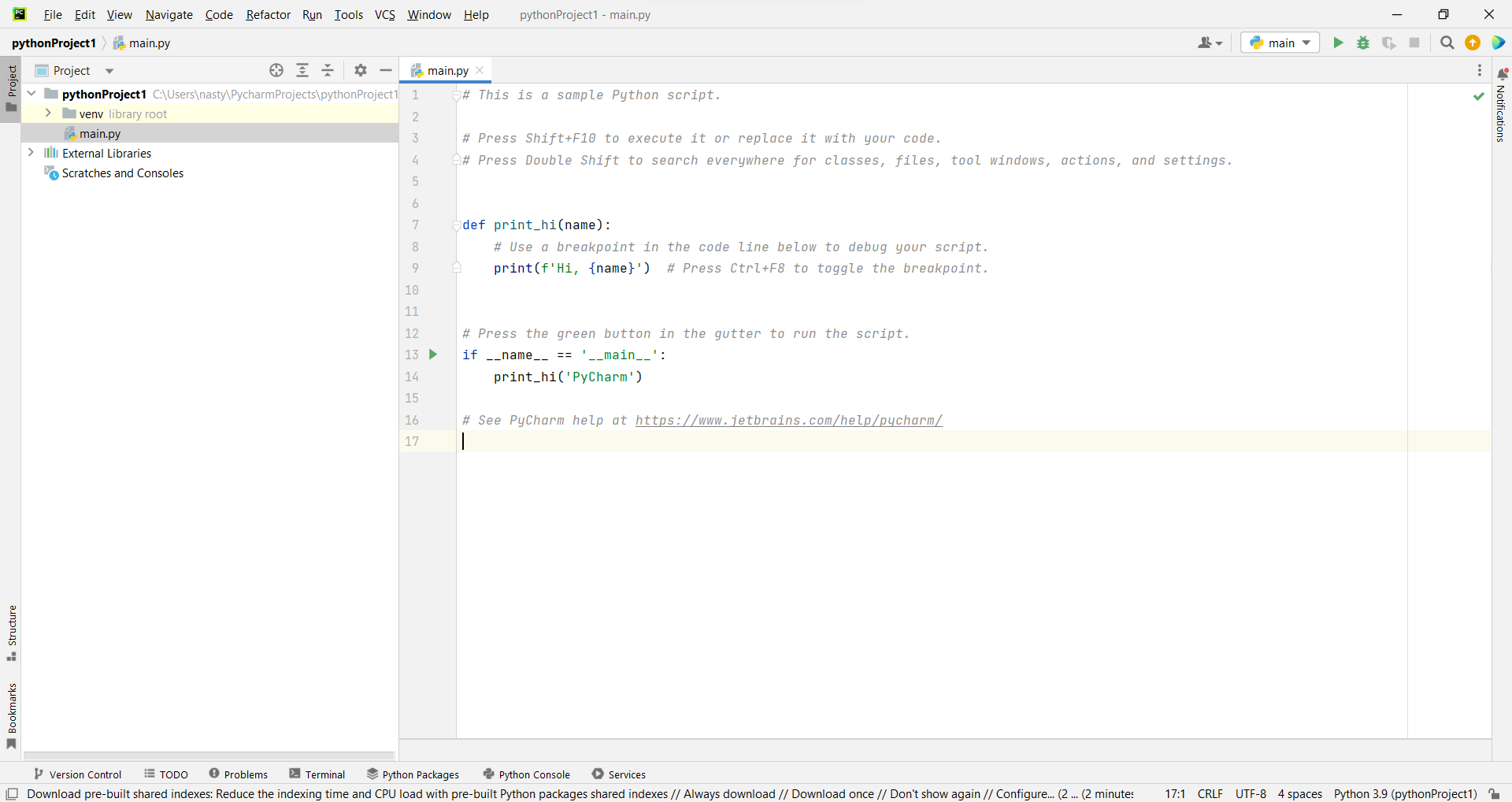


1. Выбрать местоположение вашего проекта и назвать его (обычно стоит по умолчанию);



1. Нажать кнопку Create.

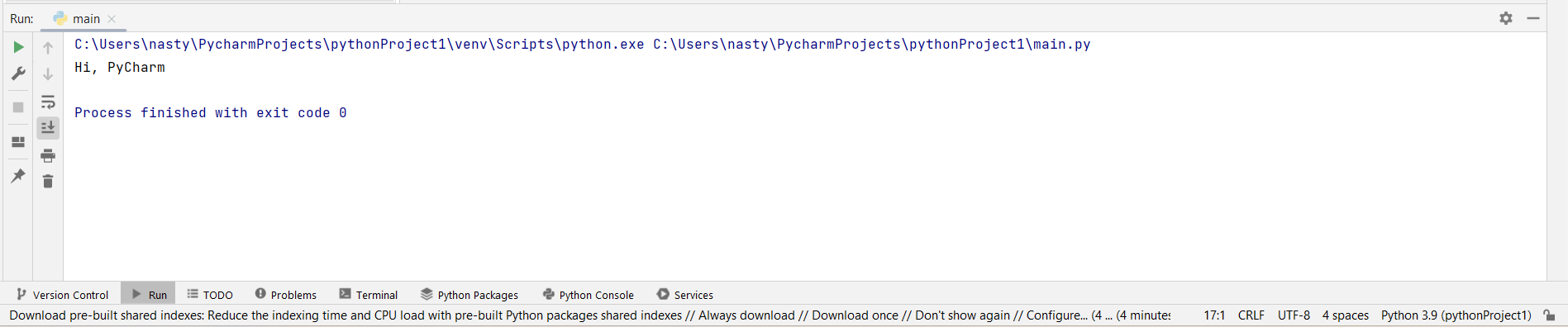
В итоге должна открыться рабочая область.



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.

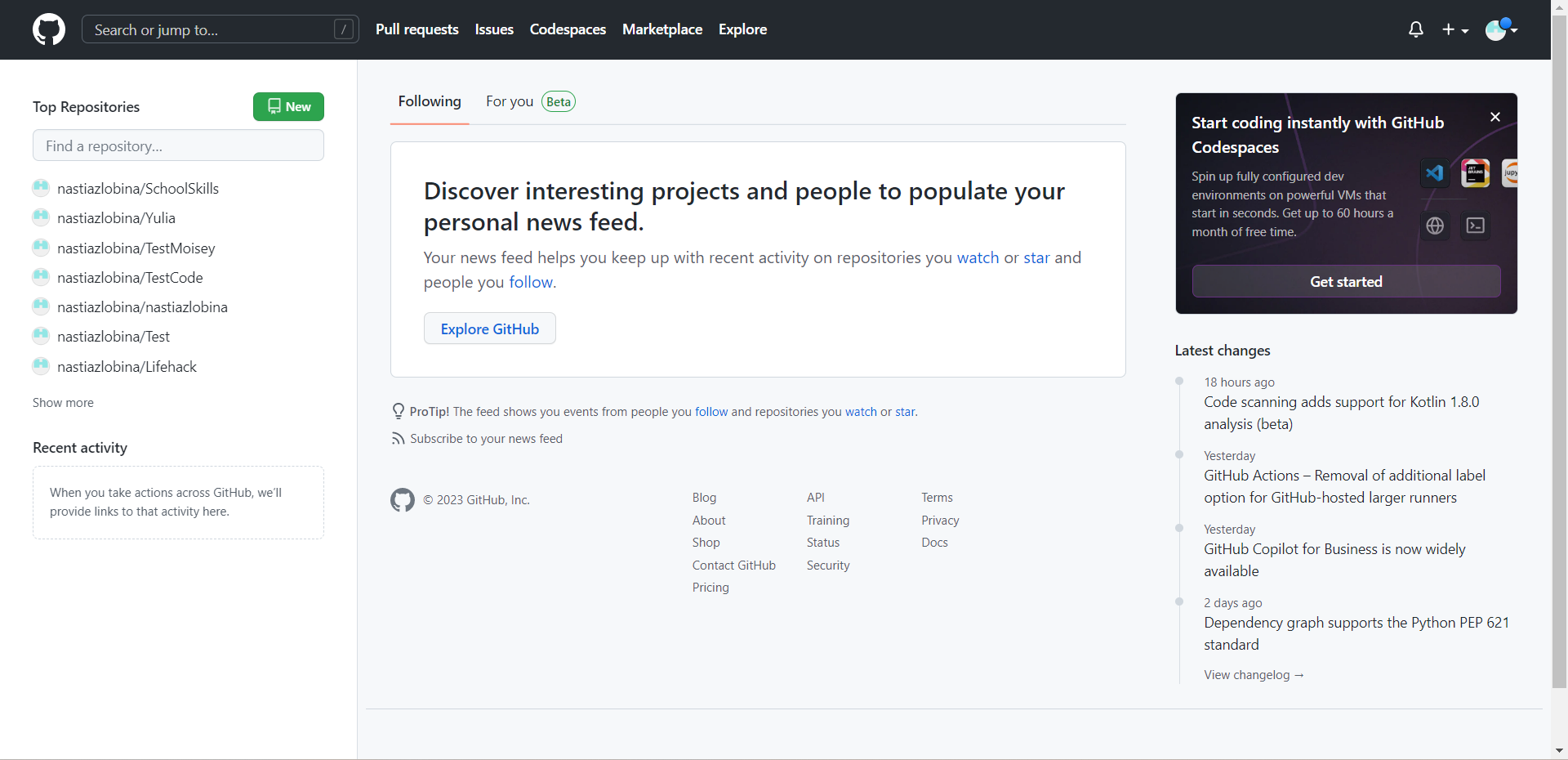
**GitHub**

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.

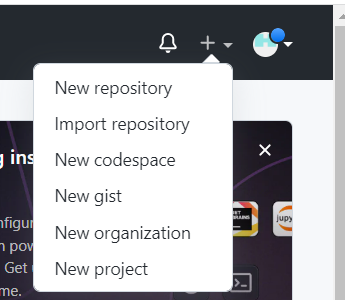
С помощью Git-a вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий.

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

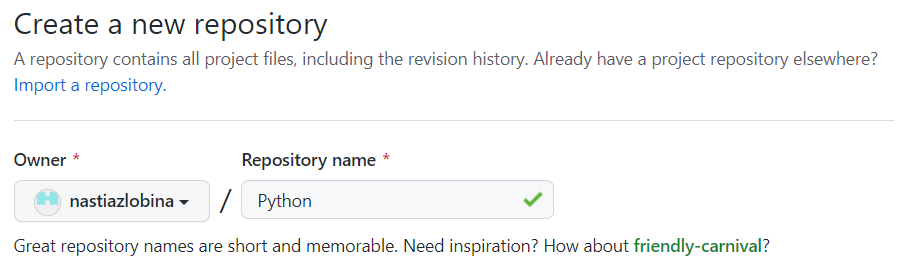
Для нашей дальнейшей работы, Вам необходимо зайти на сайт <https://github.com/> и зарегистрироваться. После регистрации, вам откроется личный кабинет.



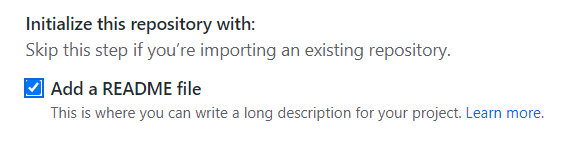
Для начала необходимо создать репозиторий, в котором будете сохранять все работы на протяжении всего курса.

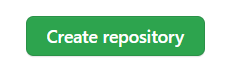


Назовите репозиторий как Python.

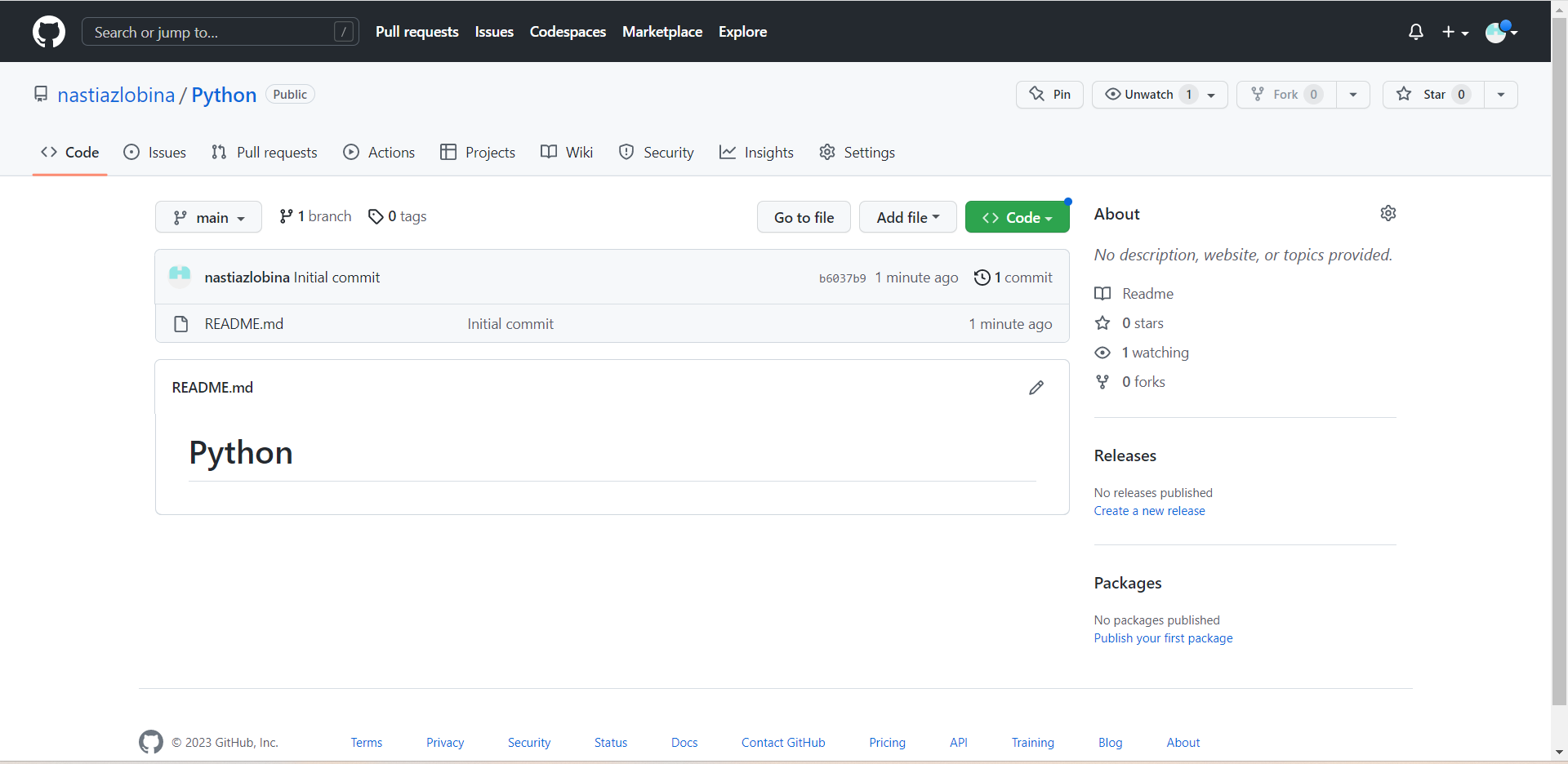


Нажмите флажок, а после создайте репозиторий.

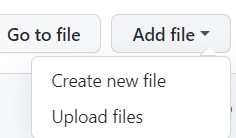




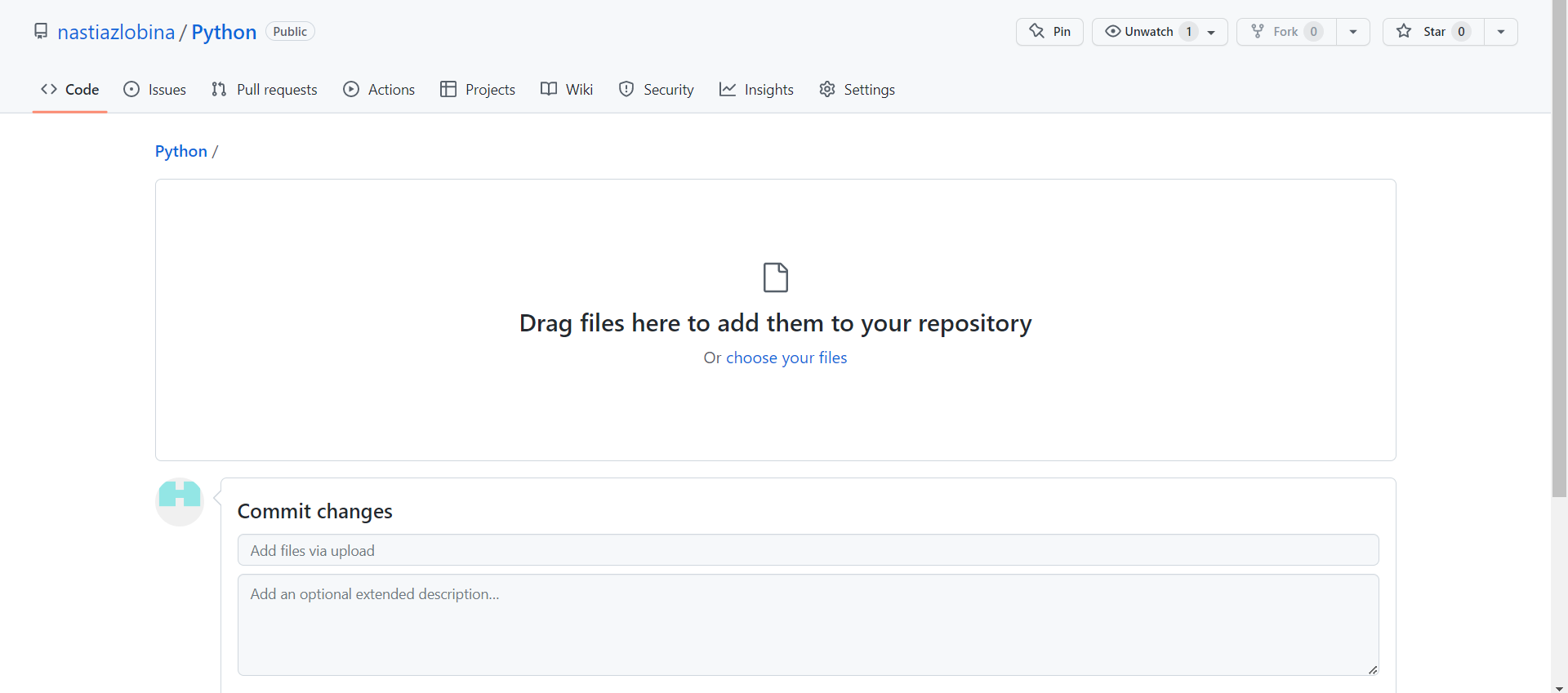
У Вас должно появиться окно с вашим репозиторием. Сюда Вы будете загружать свои работы.

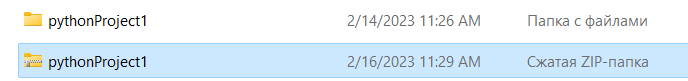


Для сохранения своей работы, необходимо нажать Add file – Upload file.



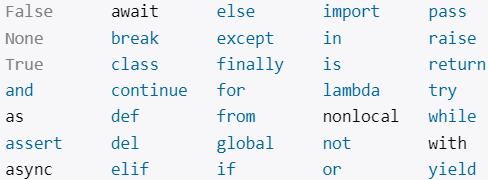
Выберите файл, который необходимо грузить. Если это программа, то она должна быть заархивирована. После того, как проект в архиве его можно загружать на Git.\



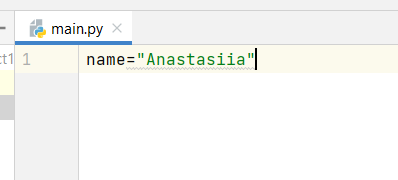


**Тема 2.1 Переменные. Ввод и вывод информации**

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**.

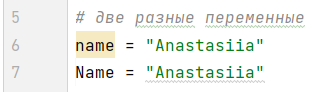
**Camel case** подразумевает, что каждое новое подслова в наименовании переменной начинается с большой буквы. Например:



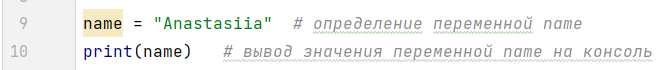
**Underscore notation** подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:



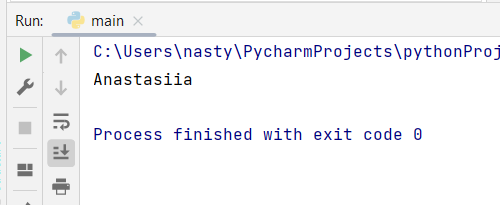
И также надо учитывать регистрозависимость, поэтому переменные **name** и **Name** будут представлять разные объекты.



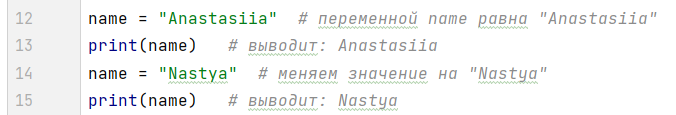
Определив переменную, мы можем использовать в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции **print**:



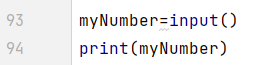
Результат работы программы:



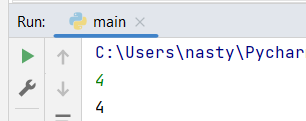
Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:



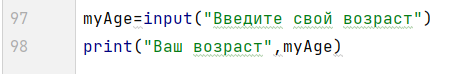
Очень часто при написании программы нужно запрашивать какую-либо информацию у пользователя. И после того, как он введет необходимые данные, их нужно сохранить в программе. Для обработки пользовательского ввода в python существует функция **input()**.

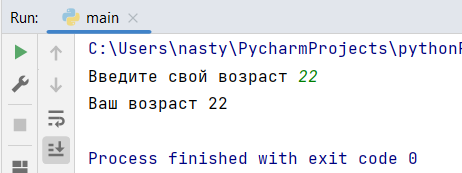


После запуска программы, если вы используете **input()**, то в консоль необходимо ввести какое-либо значение, чтобы оно записалось в переменную.



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.

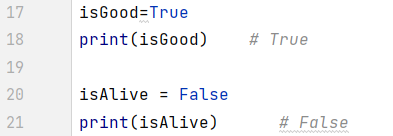
Введите значения в переменные с консоли и выведите их обратно на консоль.

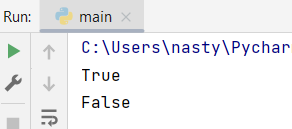
**Тема 2.2 Типы данных**

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**

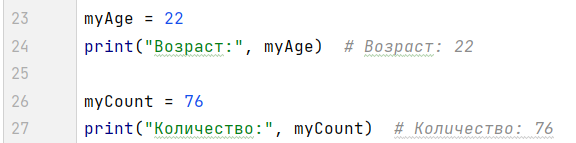
Тип **bool** представляет два логических значения: **True** (верно, истина) или **False** (неверно, ложь). Значение **True** служит для того, чтобы показать, что что-то истинно. Тогда как значение **False**, наоборот, показывает, что что-то ложно. Пример переменных данного типа:

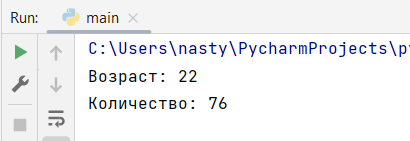




**Целые числа**

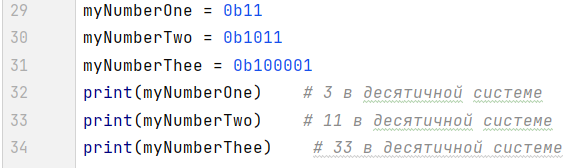
Тип int представляет целое число, например, 5, 9, 14, 55. Пример



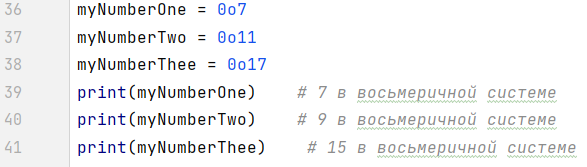


Для вывода переменных вместе с текстом, достаточно написать информацию в кавычках, поставить запятую и написать переменную, которую необходимо вывести.

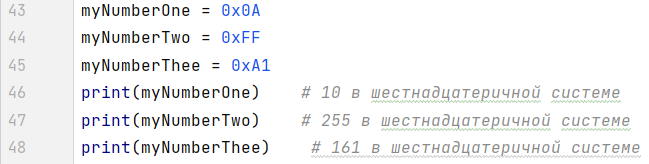
По умолчанию стандартные числа расцениваются как числа в десятичной системе. Но Python также поддерживает числа в двоичной, восьмеричной и шестнадцатеричной системах. Для указания, что число представляет двоичную систему, перед числом ставится префикс **0b**:



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



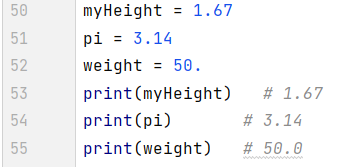
Для указания, что число представляет шестнадцатеричную систему, перед числом ставится префикс **0x**:

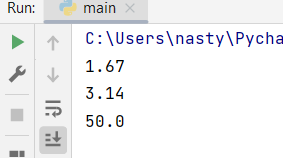


Стоит отметить, что в какой-бы системе мы не передали число в функцию **print** для вывода на консоль, оно по умолчанию будет выводиться в десятичной системе.

**Дробные числа**

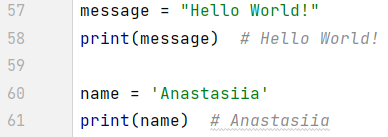
Тип float представляет число с плавающей точкой, например, 1.2 или 34.76. В качестве разделителя целой и дробной частей используется точка.

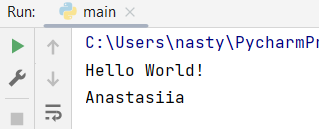




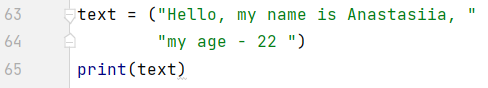
**Строки**

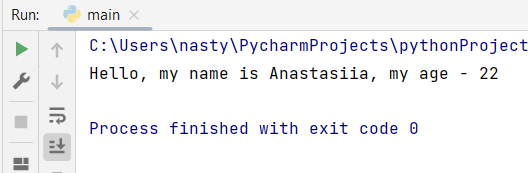
Тип **str** представляет строки. Строка представляет последовательность символов, заключенную в одинарные или двойные кавычки, например **"hello"** и **'hello'**.



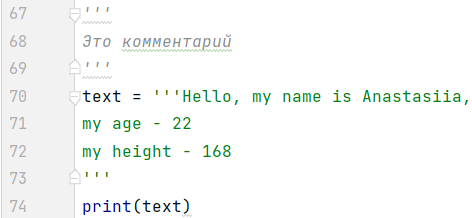


При этом если строка имеет много символов, ее можем разбить ее на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:





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.

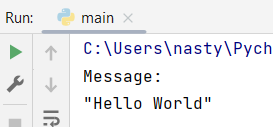
**Управляющие последовательности в строке**

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) |

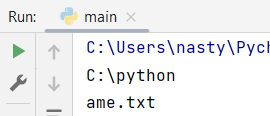
Применим несколько последовательностей:





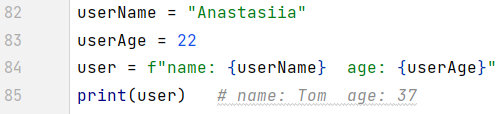
Хотя подобные последовательности могут нам помочь в некоторых делах, например, поместить в строку кавычку, сделать табуляцию, перенос на другую строку. Но они также могут и мешать. Например:



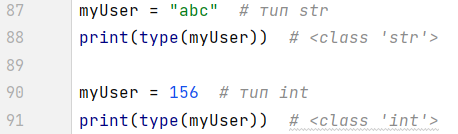


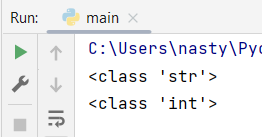
**Вставка значений в строку**

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



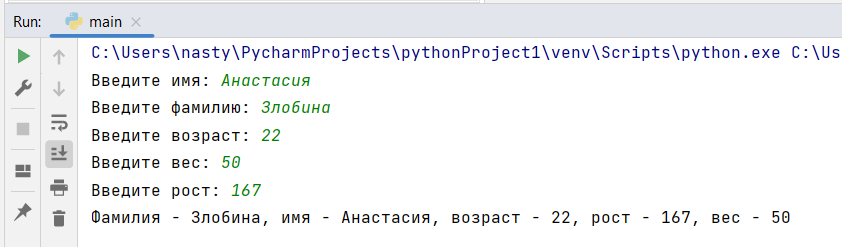
С помощью встроенной функции **type()** можно узнать текущий тип переменной:





**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.

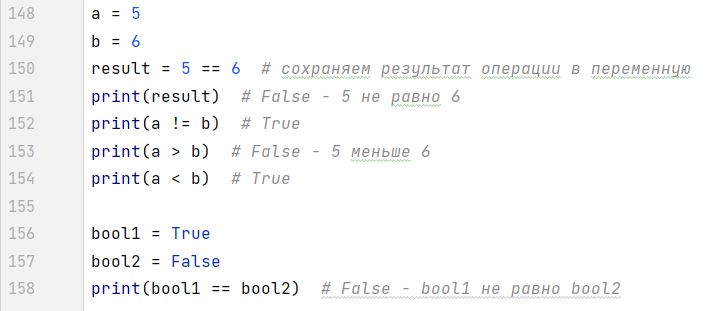
! = Возвращает True, если оба операнда НЕ равны. Иначе возвращает False.

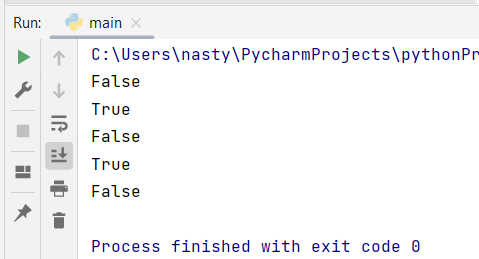
> (больше чем) Возвращает True, если первый операнд больше второго.

<(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.

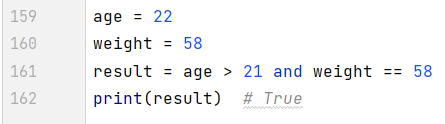
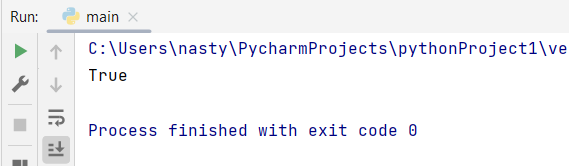




Для создания составных условных выражений применяются логические операции. В Python имеются следующие логические операторы:

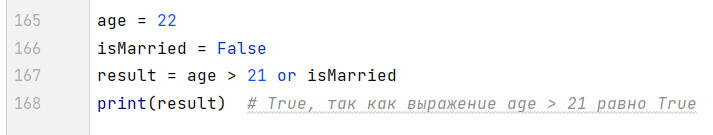
**Оператор and (логическое умножение) применяется к двум операндам:**

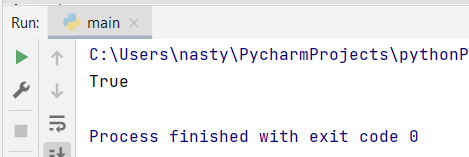
Сначала оператор **and** оценивает выражение x, и если оно равно **False**, то возвращается его значение. Если оно равно **True**, то оценивается второй операнд - y и возвращается значение y.

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

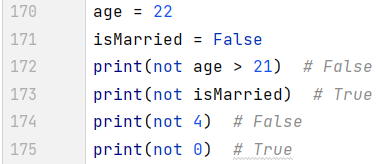
Сначала оператор or оценивает выражение x, и если оно равно True, то возвращается его значение. Если оно равно False, то оценивается второй операнд - y и возвращается значение y. Например

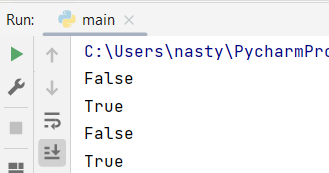




**not (логическое отрицание)**

Возвращает True, если выражение равно False



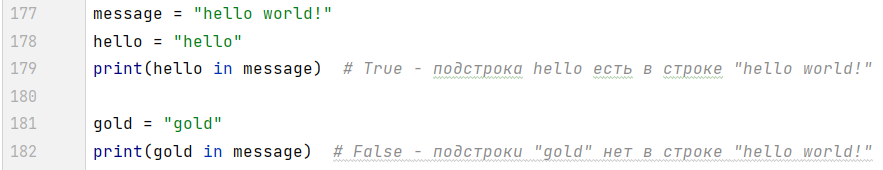
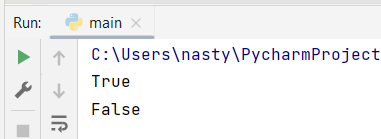


**Оператор in**

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

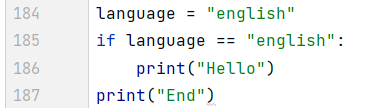
значение in набор\_значений

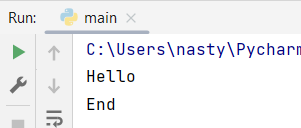
Например, строка представляет набор символов. И с помощью оператора in мы можем проверить, есть ли в ней какая-нибудь подстрока:

**Условная конструкция if**

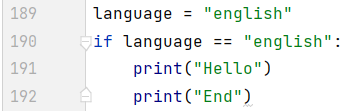
В самом простом виде после ключевого слова if идет логическое выражение. И если это логическое выражение возвращает True, то выполняется последующий блок инструкций, каждая из которых должна начинаться с новой строки и должна иметь отступы от начала выражения if (отступ желательно делать в 4 пробела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

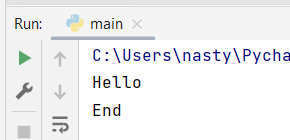




Обратите внимание в коде на последнюю строку, которая выводит сообщение "End". Она не имеет отступов от начала строки, поэтому она не принадлежит к блоку if и будет выполняться в любом случае, даже если выражение в конструкции if возвратит False.

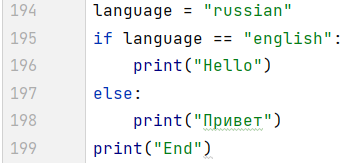
Но если бы мы поставили бы отступы, то она также принадлежала бы к конструкции if:

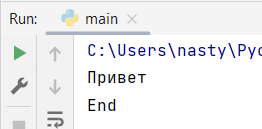




**Блок else**

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





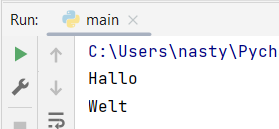
Если выражение language == "english" возвращает True, то выполняется блок if, иначе выполняется блок else. И поскольку в данном случае условие language == "english" возвращает False, то будут выполняться инструкция из блока else.

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.



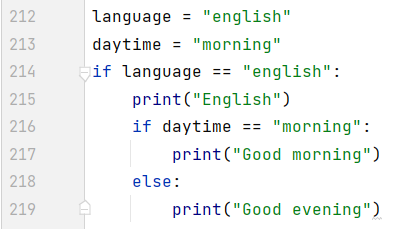


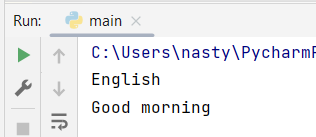
Сначала Python проверяет выражение if. Если оно равно True, то выполнениются инструкции из блока if. Если это условие возвращает False, то Python проверяет выражение из elif.

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.