**Тема занятия №01: Введение в программирование на Python**

**Введение**

Программу можно представить, как набор последовательных команд, то есть алгоритм, для объекта, то есть исполнителя, который должен их выполнить для достижения определенной цели.

Так можно условно запрограммировать человека, составив для него к примеру инструкцию "как приготовить оладьи", а он начнет четко ей следовать. При этом инструкция, она же программа, для человека будет написана на так называемом естественном языке, например, русском или английском.

Все же программируют не людей, а вычислительные машины, используя при этом специальные языки. Необходимость в особых языках связана с тем, что машины не в состоянии "понимать" наши, то есть человеческие естественные для нас языки. Инструкции для машин пишут на языках программирования, которые характеризуются формальностью, то есть синтаксической однозначностью (например, в них нельзя менять местами определенные слова) и ограниченностью (имеют строго определенный набор слов и символов).

**1.1 Основные этапы исторического развития языков программирования**

Первые программы писались на машинном языке, так как для ЭВМ того времени еще не существовало развитого программного обеспечения, а машинный язык – это единственный способ взаимодействия с аппаратным обеспечением компьютера, так называемым "железом".

Каждую команду машинного языка непосредственно выполняет то или иное электронное устройство. Данные и команды записывали в цифровом виде, например, в шестнадцатеричной или двоичной системах счисления. Человеку воспринимать программу на таком языке сложно. Кроме того, даже небольшая программа состояла из множества строк кода. Ситуация осложнялась еще и тем, что каждая вычислительная машина понимает лишь свой машинный язык.

Людям, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами, а не цифрами привело к появлению ассемблеров. Это языки, в которых вместо численного обозначения команд и областей памяти используются словесно-буквенные.

При этом появляется проблема: машина не в состоянии понимать слова. Необходим какой-нибудь переводчик на ее родной машинный язык. Поэтому, начиная со времен ассемблеров, под каждый язык программирования создаются трансляторы – специальные программы, преобразующие программный код с языка программирования в машинный код. Ассемблеры на сегодняшний день продолжают использоваться. В системном программировании с их помощью создаются низкоуровневые интерфейсы операционных систем, компоненты драйверов.

После ассемблеров наступил расцвет языков так называемого высокого уровня. Для них потребовалось разрабатывать более сложные трансляторы, так как языки высокого уровня куда больше удобны для человека, чем для вычислительной машины.

В отличие от ассемблеров, которые остаются привязанными к своим типам машин, языки высоко уровня обладают переносимостью. Это значит, что, написав один раз программу, программист без последующего редактирования может выполнить ее на любом компьютере, если на нем установлен соответствующий транслятор. Программа-транслятор для данной ЭВМ при трансляции исходного кода сама адаптирует его под эту ЭВМ.

Следующим значимым шагом было появление объектно-ориентированных языков, что в первую очередь связано с усложнением разрабатываемых программ. С помощью таких языков программист как бы управляет виртуальными объектами. Мыслить в рамках объектов-сущностей, описывать их взаимодействие, обобщать объекты в классы и устанавливать между ними наследственные связи, – все это делает программу по-своему похожей на реальный мир, на то, как его воспринимает человек.

На сегодняшний день в большинстве случаев реализация крупных проектов осуществляется с помощью объектно-ориентированных возможностей языков. Хотя существуют и другие современные парадигмы программирования, поддерживаемые другими или теми же языками.

**Разнообразие языков программирования**

В настоящее время существует множество различающихся и похожих между собой языков программирования. Причина такого явления становится понятна, если представить, то количество и разнообразие задач, которые на сегодняшний день решается с помощью вычислительной техники. Для решения разных задач требуются разные инструменты, то есть разные языки и подходы к программированию.

Разработка новых языков программирования, обладающий теми или иными преимуществами, велась как в прошлом, так и ведется сейчас. Эволюционируют, подстраиваясь под запросы нового времени, и старые языки программирования.

Все многообразие языков можно классифицировать по разным критериям. Например, по типу решаемых задач (языки системного или прикладного назначения, языки для web-разработки, организации баз данных, разработки мобильных приложений). Среди наиболее популярных на сегодняшний день можно отметить Java, C, C++, C#, JavaScript, PHP, в том числе Python.

**1.2 Трансляция**

Ранее было сказано, что для перевода кода с языка программирования высокого уровня на машинный язык требуется специальная программа – транслятор.

Заложенный в транслятор алгоритм такого перевода сложен. При этом существует два основных способа трансляции — компиляция программы или ее интерпретация.

При компиляции весь исходный программный код (тот, который пишет программист) сразу переводится в машинный. Создается так называемый отдельный исполняемый файл, который никак не связан с исходным кодом. Выполнение исполняемого файла обеспечивается операционной системой (ОС). После того как получен исполняемый файл, для его чтения транслятор уже не нужен.

При интерпретации выполнение кода происходит последовательно (условно можно сказать, строка за строкой). Грубо говоря, операционная система взаимодействует с интерпретатором, а не с файлом, содержащим программный код. Интерпретатор же, прочитав очередную часть исходного кода, переводит его в машинный (или не совсем машинный, но "понятный" для ОС) и "отдает" его ОС. ОС исполняет этот код и ждет следующей "подачки" от интерпретатора. Питон именно такой язык. Он интерпретируемый язык программирования.

Выполнение откомпилированной программы происходит быстрее, так как она представляет собой готовый машинный код. Однако на современных компьютерах снижение скорости выполнения при интерпретации обычно не заметно. Кроме того, интерпретируемые языки обладают рядом преимуществ, среди которых отсутствие подготовительных действий для исполнения программы, что может быть важным для тех, кто только начинает изучать программирование.

Язык программирования Python был создан к 1991 году голландцем Гвидо ван Россумом.

Свое имя – Пайтон (или Питон) – получил от названия телесериала, а не пресмыкающегося.

После того, как Россум разработал язык, он выложил его в Интернет, где сообщество программистов присоединилось к его улучшению.

**1.3 Основные особенности языка**

**Python** – интерпретируемый язык программирования. Это значит, что исходный код частями преобразуется в машинный в процессе его чтения специальной программой – интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на нем легче, чем на других языках программирования, так как в Питоне мало используются такие вспомогательные синтаксические элементы как скобки, точки с запятыми. С другой стороны, правила языка заставляют программистов делать отступы для обозначения вложенных конструкций. Понятно, что хорошо оформленный текст с малым количеством отвлекающих элементов читать и понимать легче.

**Python** – это полноценный во многом универсальный язык программирования, используемый в различных сферах. Основная, но не единственная, поддерживаемая им парадигма, – объектно-ориентированное программирование. Однако в данном курсе мы только упомянем об объектах, а будем изучать структурное программирование, так как оно является базой. Без знания основных типов данных, ветвлений, циклов, функций нет смысла изучать более сложные парадигмы, так как в них все это используется.

Интерпретаторы Python распространяется свободно на основании лицензии подобной GNU General Public License.

**1.4 Дзен Питона**

Если интерпретатору Питона дать команду import this, то выведется так называемый "Дзен Питона", иллюстрирующий идеологию и особенности данного языка. Понимание смысла этих постулатов в приложении к программированию придет тогда, когда вы освоите язык в полной мере и приобретете опыт практического программирования.

Beautiful is better than ugly. Красивое лучше уродливого.

Explicit is better than implicit. Явное лучше неявного.

Simple is better than complex. Простое лучше сложного.

Complex is better than complicated. Сложное лучше усложнённого.

Flat is better than nested. Плоское лучше вложенного.

Sparse is better than dense. Разрежённое лучше плотного.

Readability counts. Удобочитаемость важна.

Special cases aren't special enough to break the rules. Частные случаи не настолько существенны, чтобы нарушать правила.

Although practicality beats purity. Однако практичность важнее чистоты.

Errors should never pass silently. Ошибки никогда не должны замалчиваться.

Unless explicitly silenced. За исключением замалчивания, которое задано явно.

In the face of ambiguity, refuse the temptation to guess. Перед лицом неоднозначности сопротивляйтесь искушению угадать.

There should be one — and preferably only one — obvious way to do it. Должен существовать один — и, желательно, только один — очевидный способ сделать это.

Although that way may not be obvious at first unless you're Dutch. Хотя он может быть с первого взгляда не очевиден, если ты не голландец.

Now is better than never. Сейчас лучше, чем никогда.

Although never is often better than \*right\* now. Однако, никогда чаще лучше, чем прямо сейчас.

If the implementation is hard to explain, it's a bad idea. Если реализацию сложно объяснить — это плохая идея.

If the implementation is easy to explain, it may be a good idea. Если реализацию легко объяснить — это может быть хорошая идея.

Namespaces are one honking great idea — let's do more of those! Пространства имён — прекрасная идея, давайте делать их больше!

**1.5 Интерактивный режим**

Грубо говоря, интерпретатор выполняет команды построчно. Пишешь строку, нажимаешь Enter, интерпретатор выполняет ее, наблюдаешь результат.

Это удобно, когда изучаешь особенности языка или тестируешь какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, пришлось бы сначала создать файл с кодом на исходном языке программирования, затем передать его компилятору, получить от него исполняемый файл и только потом выполнить программу и оценить результат. К счастью, даже в случае с компилируемыми языками все эти действия выполняет среда разработки, что упрощает жизнь программиста.

В операционных системах на базе ядра Linux можно программировать на Python в интерактивном режиме с помощью приложения «Терминал», в котором работает командная оболочка Bash. Здесь, чтобы запустить интерпретатор, надо выполнить команду python3.

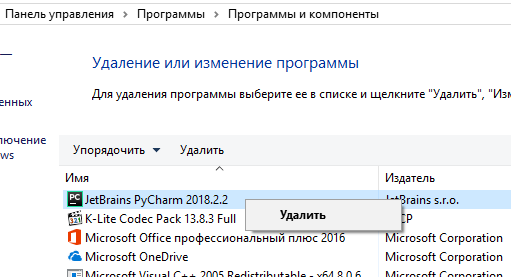
**1.6 Установка программного обеспечения**

Установка Anaconda, Python и всего прочего, «чтоб работало».   
Сначала удаляем все старые версии Python, PyCharm, Anaconda, чтоб все начать «с нуля».

Пуск – Установка и удаление программ (при работе в Windows 7, 8, 10 достаточно нажать кнопку «Пуск» и начать набирать название нужной программы)

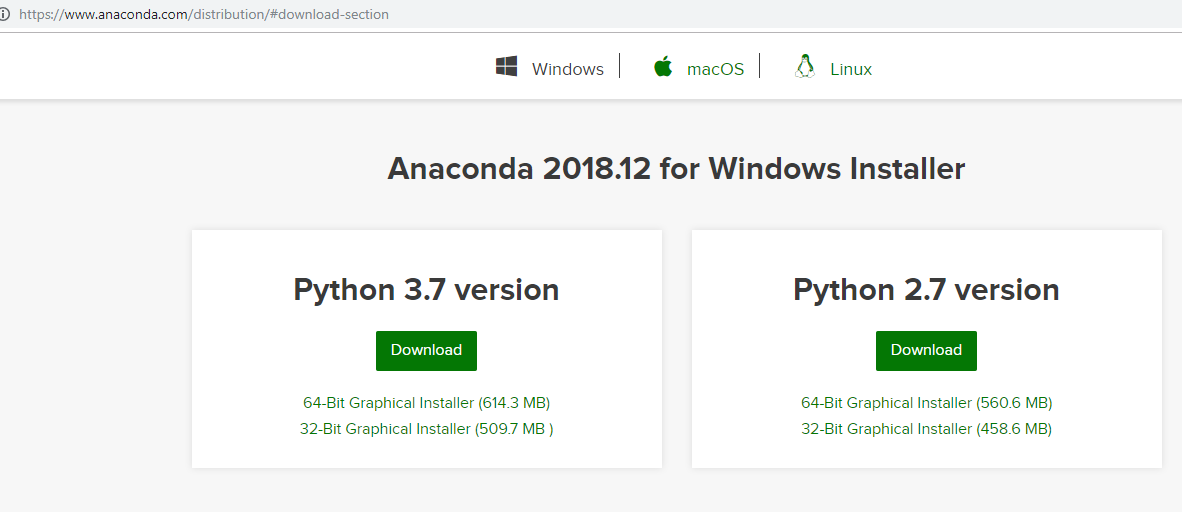
Либо Пуск - **appwiz.cpl (копируем, нажимаем Пуск, вставляем)**

**Удаляем все, что относится к Python (сам Python, все старые версии, PyCharm, Anaconda).**



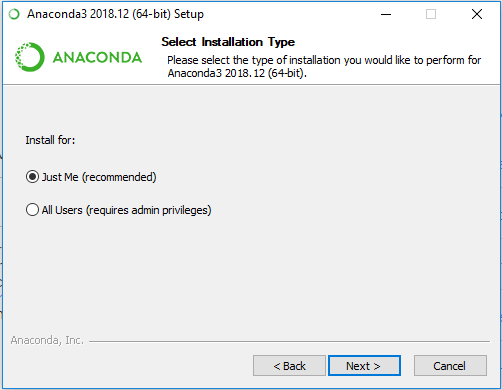
**Установка Anaconda:**

C сайта <https://www.anaconda.com/distribution/> скачиваем и устанавливаем последнюю версию Анаконды. Python версии 3, естественно.

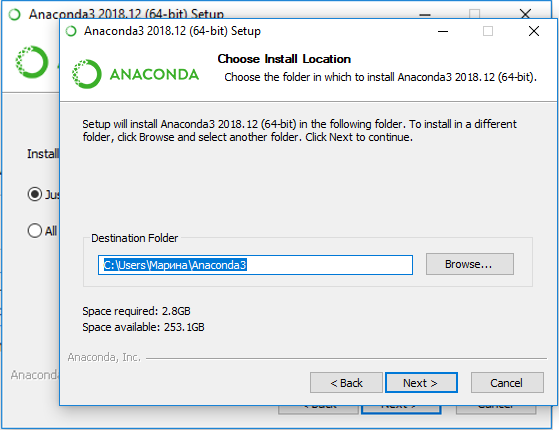


Дистрибутив достаточно большой (около 600Мб).

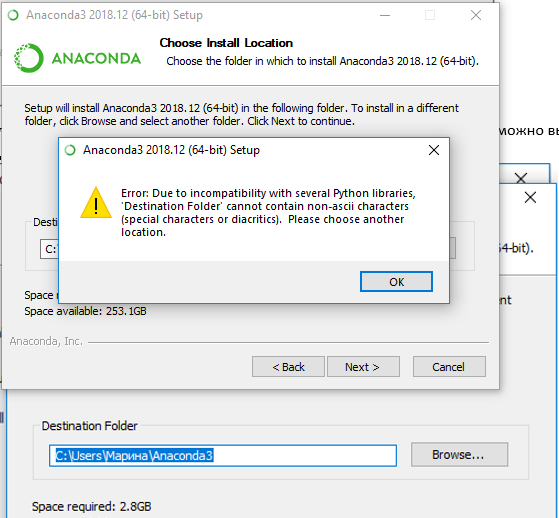
При установке выбираем вариант установки (только для себя или для всех пользователей). Рекомендованный вариант – только для себя. Но тут мы сталкиваемся с рядом проблем.



В этом случае Анаконда установится по умолчанию в вашу папку пользователя (можно выбрать и любую другую):



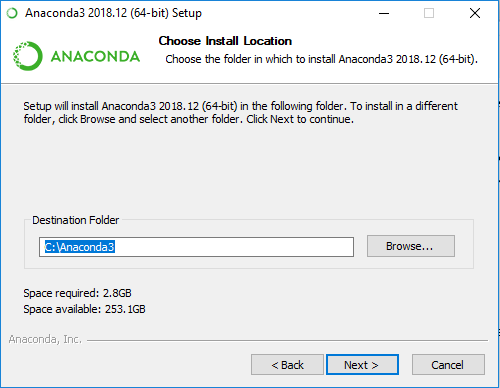
Ой нет, установка в папки с кириллическими названиями не удается:



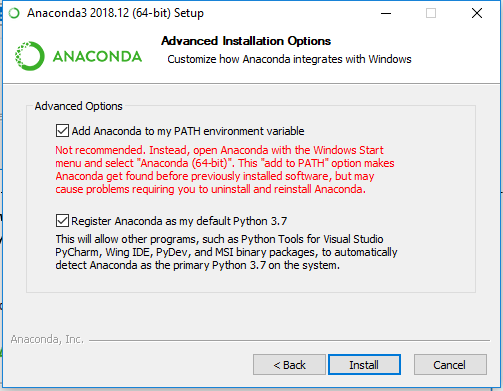
Установить в пользовательскую папку, если вы назвали пользователя кириллицей, не выйдет. Не поступайте так больше, придумывайте имена латиницей )))))

Ок, выбираем любую папку, в которую у вас есть доступ. Или все же устанавливаем «для всех пользователей», если у вас все в порядке с правами администратора. На Диск С обычно пользователю windows доступ ограничен, поэтому папку для программы придется создать заранее. Если у вас есть второй диск в системе, лучше выбрать его.

Я создаю папку C:\Anaconda3 и выбираю ее для установки.



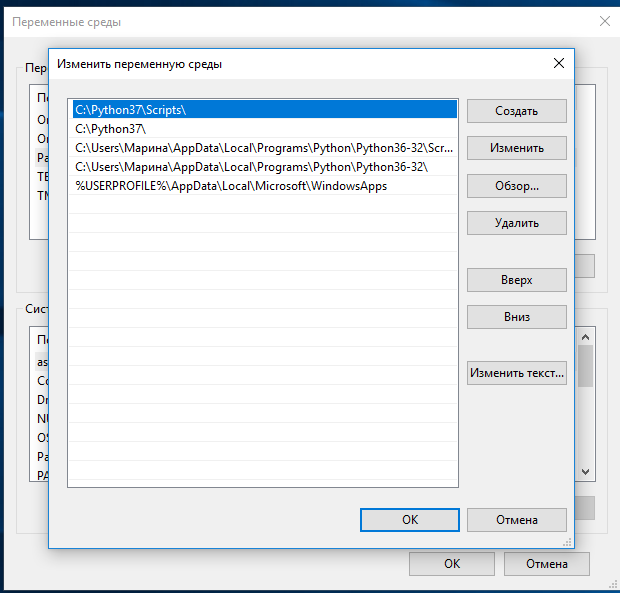
Добавлять Anaconda в переменную PATH не рекомендуется, т.к. написано, что это может вызвать проблемы при удалении и переустановке программы. Все-таки я сделаю это, но нужно не забыть потом руками чистить переменную PATH от старого «мусора». Проверяем, что стоит галочка «Register Anaconda as my default Python 3.7»



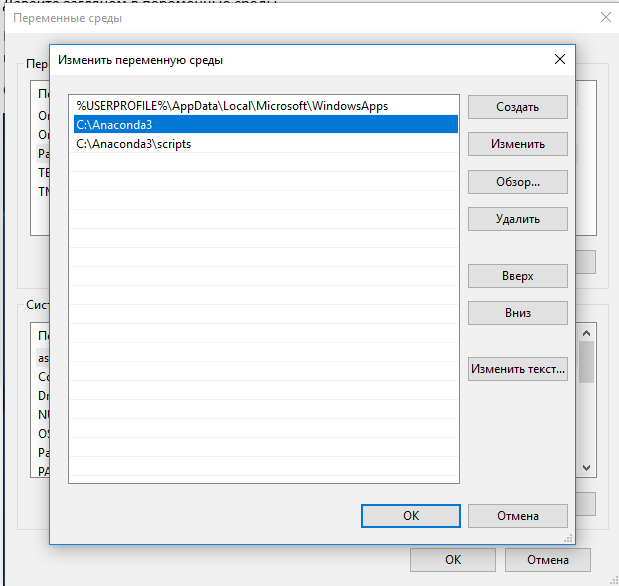
Давайте заглянем в переменные среды.

Пуск – (начинаем набирать «переменных») – Изменение системных переменных среды. Щелкаем по Path.

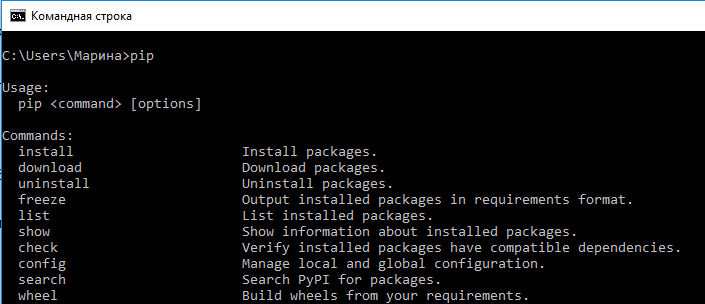
Смотрите, сколько у меня прописано старых версий Python, которых давно уже нет.



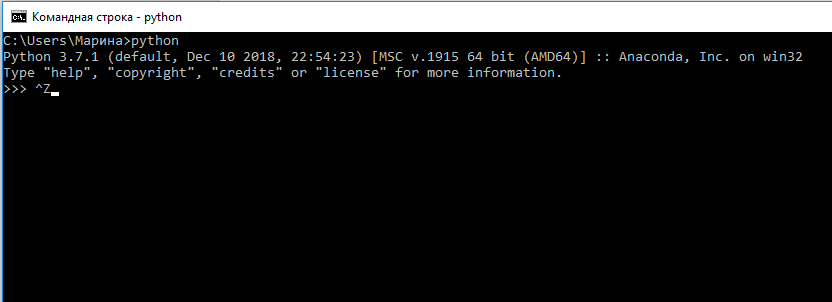
Удаляем все, относящееся к Python, оставляем только путь к свежепоставленной Анаконде. У меня автоматически Анаконда автоматически не добавилась, несмотря на поставленную галочку. Поэтому добавляем пути к Анаконде и папке Scripts (где находится pip).



Теперь при запуске командной строки (Пуск – cmd) мы можем вызвать команду pip. Если все сделано верно, она запустится и выдаст справку по использованию.

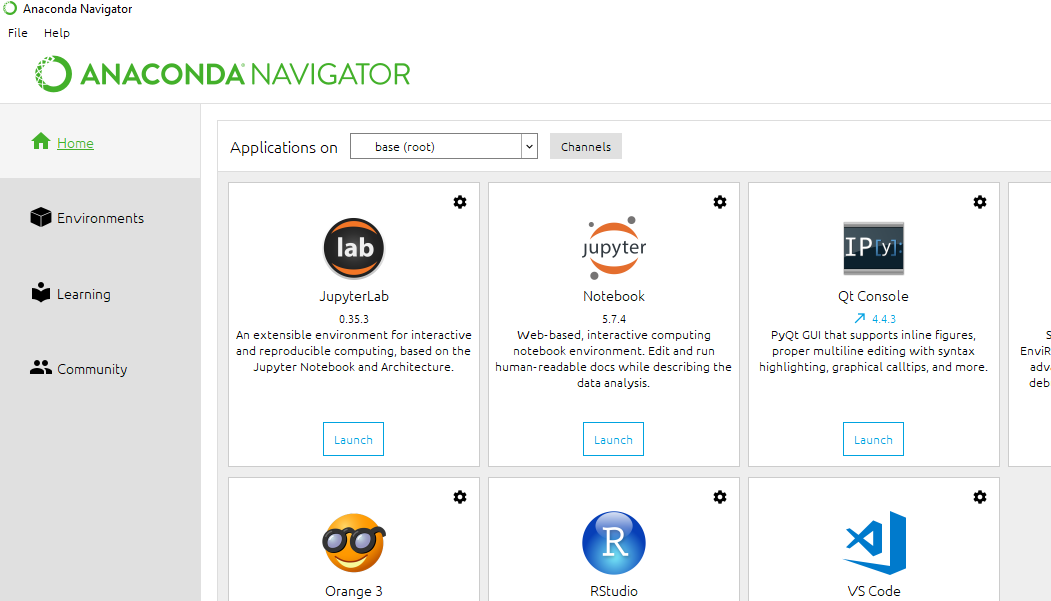


Если набрать команду python, запустится командная строка Python. (Выход из нее – Ctrl+D)



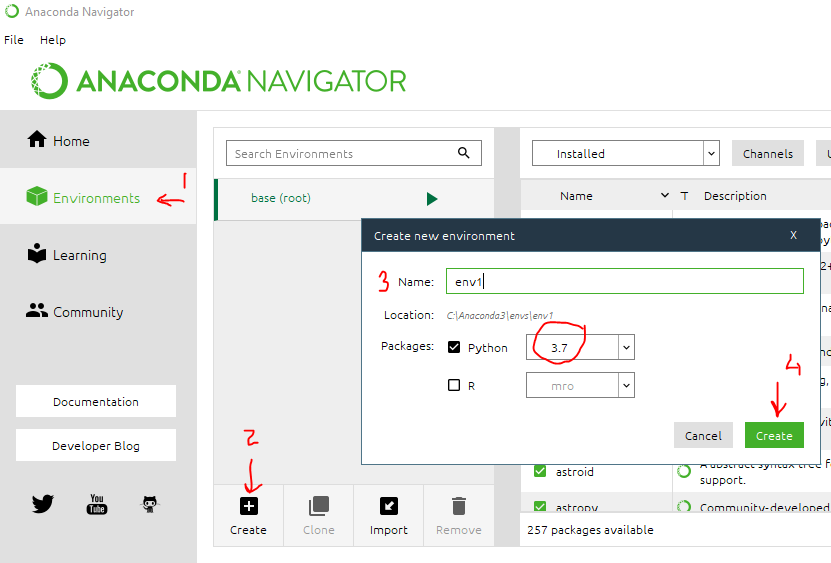
Ок, все установлено верно, Python в системе виден, никаких других питонов не установлено, в переменных среды ничего лишнего.

Запускаем Anaconda Navigator (Через Пуск).



Первым делом создадим новое виртуальное окружение и будем работать в нем.

Environments – Create – Выбираем название на английском (напр. env1), версию python и нажимаем Create.



Дальше мы можем проводить любые эксперименты, устанавливать любые библиотеки в этом локальном окружении.

Ждем, пока все установится.

Возвращаемся в Home, нажимаем Install у Jupyter Notebook, снова ждем.

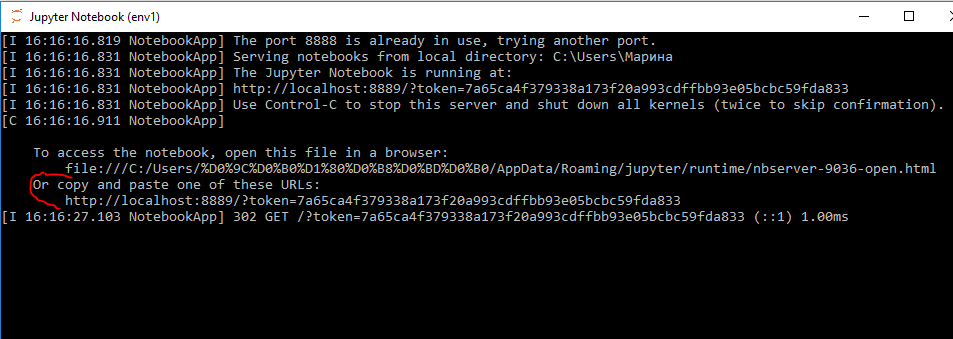
Из всего богатства нам пока нужен только Jupyter Notebook.

Запускаем его, он откроется в браузере.

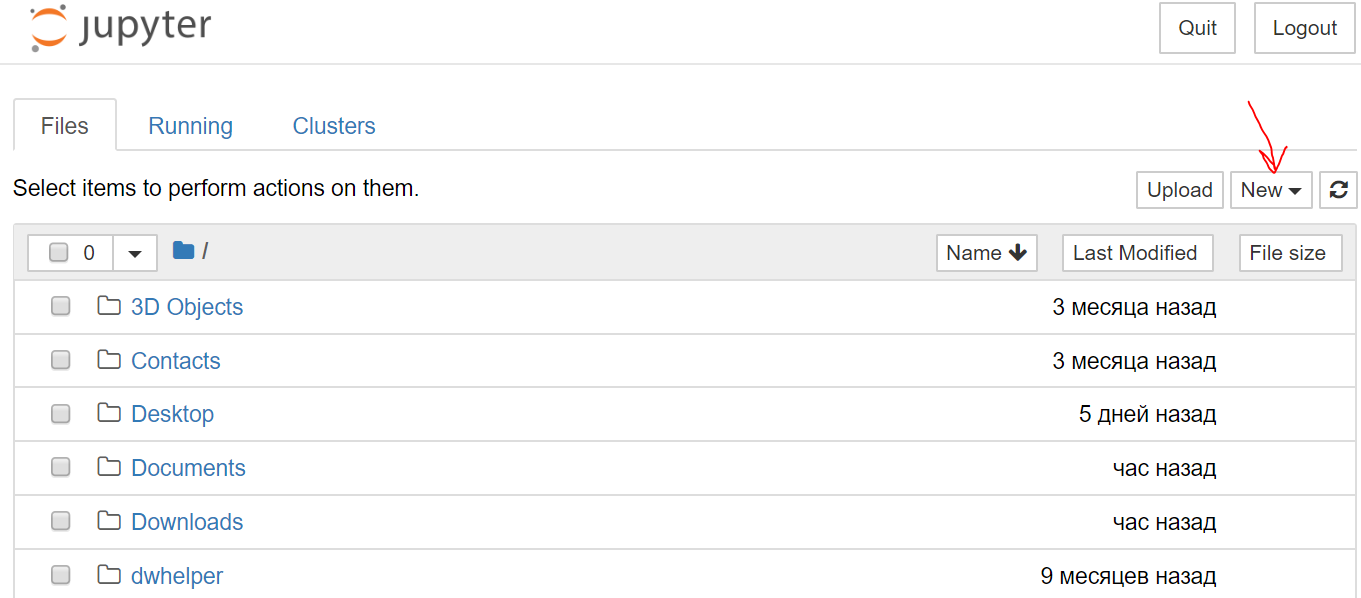
Упс… У всех запускается, а у меня на компьютере не хочет автоматически открываться в браузере. Если у вас все хорошо, пропускайте этот пункт. А если ваш комп тоже решил проигнорировать, то в дальнейшем запускаем jupyter так.

Пуск – Jupyter Notebook (предложил даже два варианта, второй во вновь созданном окружении env1).

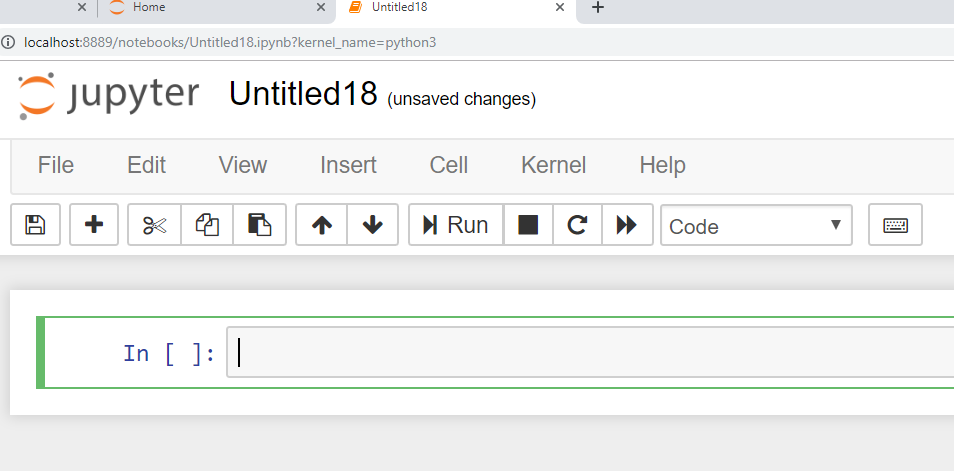
Открывается окно терминала:



Копируем ссылку и вставляем в окно браузера вручную. Надеюсь, никому больше это не пригодится ☺



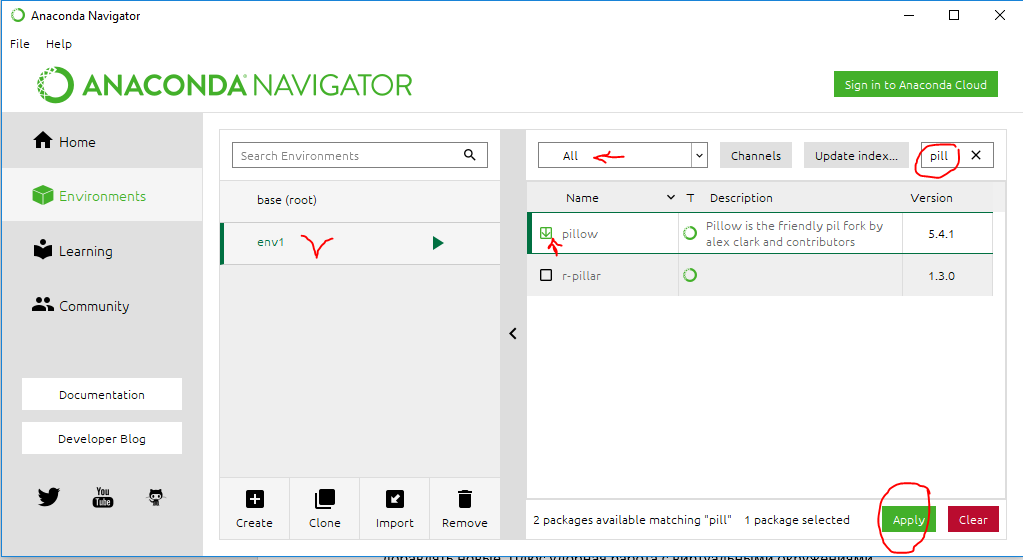
Нажимаем New, Python3 и наслаждаемся открывшимся окном Jupyter.



(Кстати, Jupyter Notebook можно установить просто через обычный python и pip, не устанавливая Анаконду). Но в Анаконде есть много предустановленных пакетов, а также удобный способ добавлять новые. Плюс удобная работа с виртуальными окружениями.

**Итак, как устанавливать библиотеки.**

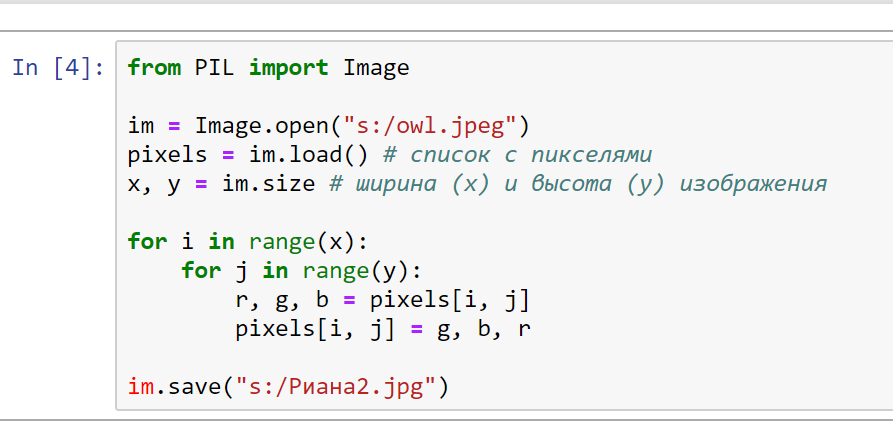
Библиотека Pillow уже есть в Анаконде, но в нашем новом окружении env1 нужно поставить ее заново.



Выбираем «Все пакеты» (All), начинаем набирать название pillow, ставим галочку и выбираем Apply. Нам предлагают установить 10 необходимых пакетов, соглашаемся.

Проверяем работу на примере из учебника. Сохраняем изображение с совой на диск (у меня: s:\\owl.jpeg) и немного исправляем программу. По умолчанию Jupyter будет искать и сохранять изображение в папке по умолчанию (в моем случае это C:\\Users\Марина).

Поэтому для того, чтоб работы с другими папками, прописываем путь полностью. Используем прямой одиночный слэш! Обратный слэш служит для спецсимволов )).



Таким образом устанавливаем все нужные библиотеки.

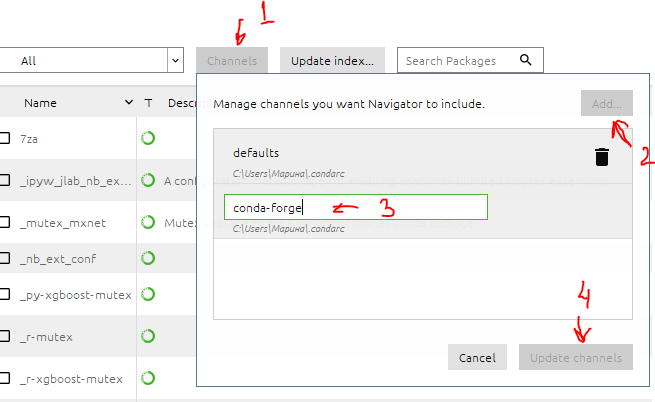
Но…

Некоторые библиотеки Anaconda не находит по умолчанию. Например, нужные нам python-docx и python-pptx.

Добавляем канал conda-forge.

Channels – Add – Пишем «conda-forge» и нажимаем Enter – Updates channels.

Все, python-docx и python-pptx теперь есть в списке.

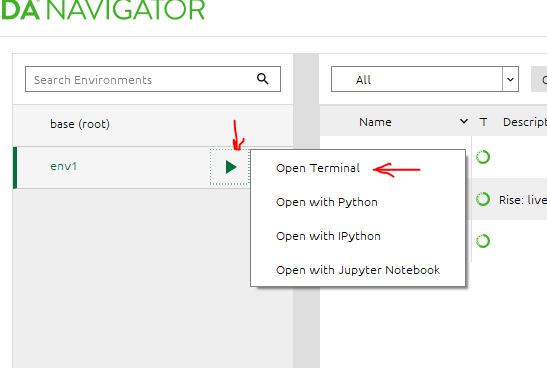


Еще один небольшой фокус.

В Jupyter Notebook можно создавать презентации.

Нам нужна библиотека rise (включаем ее, как уже научились).

Дальше запускаем терминал (для нашего env1, если мы работаем в виртуальном окружении) через вкладку Environments:



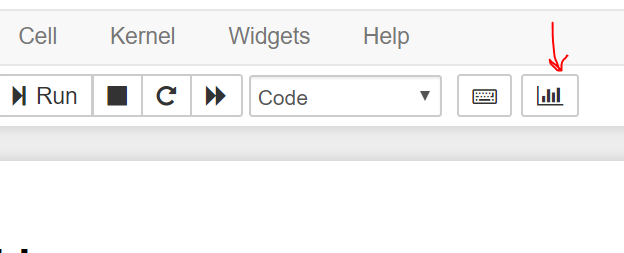
И выполняем последовательно в терминале 2 команды (copy-past):

jupyter-nbextension install rise --py --sys-prefix

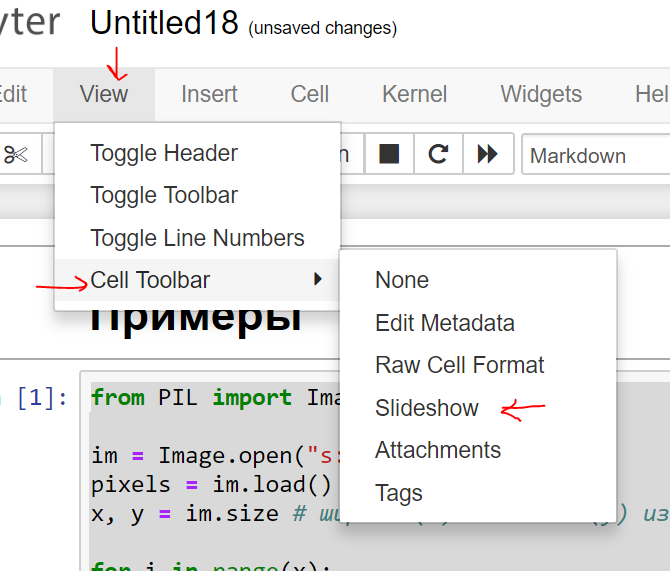
jupyter-nbextension enable rise --py --sys-prefix

Перезапускаем Jupyter (если он открыт, то достаточно нажать кнопку Restart kernel на панели инструментов).

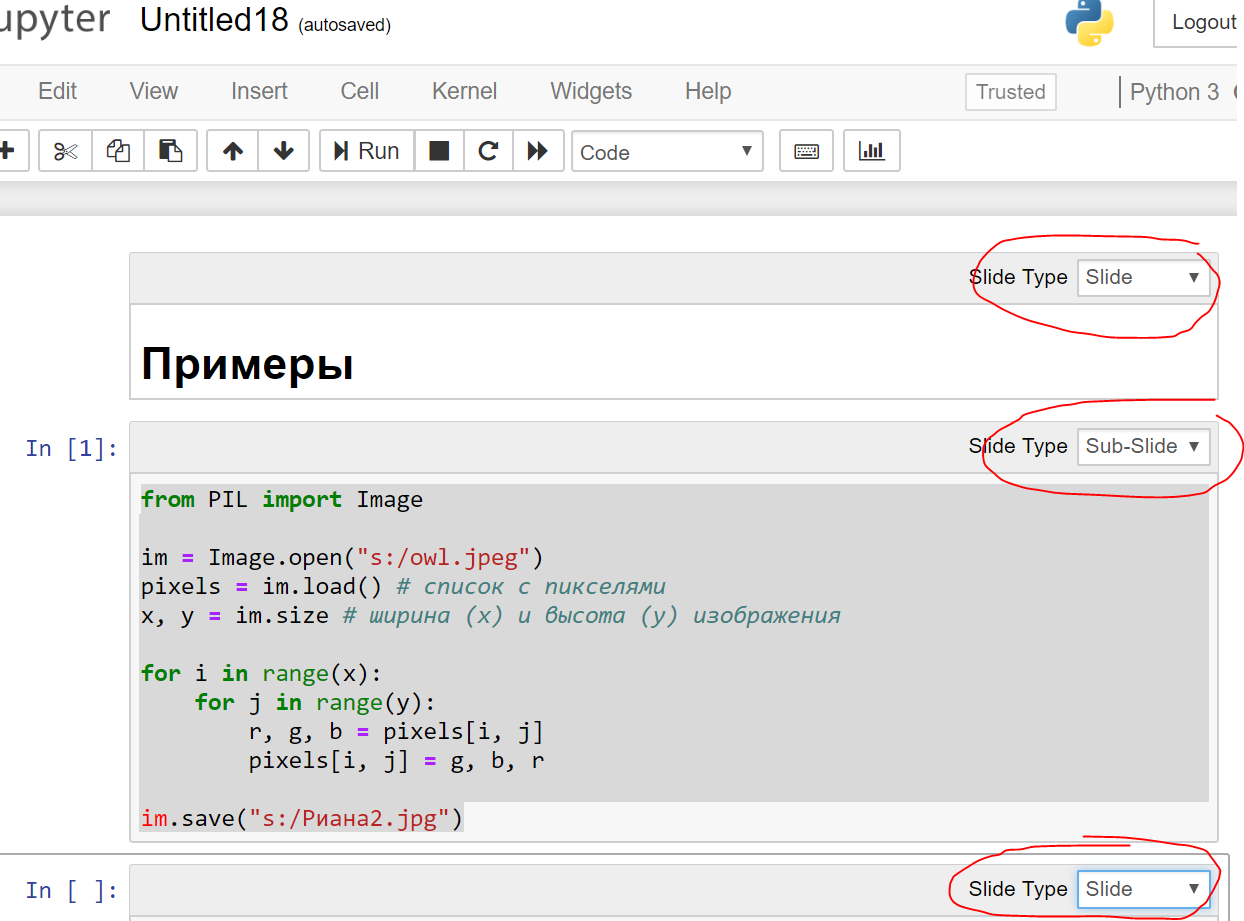
Если все сделали правильно, то появилась новая кнопка:



Нажимаем View – Cell Toolbar – Slideshow



И приступаем к разметке слайдов.



Затем нажимаем кнопку Слайдшоу и наслаждаемся эффектом.

Полное описание RISE (горячие клавиши и т.п.)

<https://rise.readthedocs.io/en/docs_hot_fixes/>

Переходим к установке PyCharm.

**1.7 Установка PyCharm**

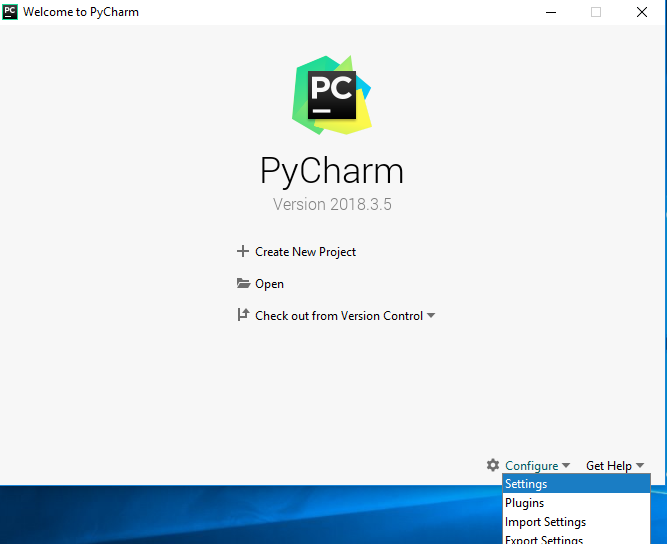
С сайта <https://www.jetbrains.com/pycharm/download/#section=windows> скачиваем PyCharm (версия Community).

Установка и настройка PyCharm подробно разобрана в видео от Яндекс-лицея из первых уроков.

Но мы хотим «совместить» нашу уже установленную Анаконду со всеми модулями и PyCharm. Тогда не придется ставить модули дважды и плодить виртуальные окружения.

Кроме того, при отсутствии прав администратора установка библиотек через pycharm (и pip) не всегда работает. Спасением может стать подключение виртуального окружения Анаконды, где уже установлены все пакеты.

**Запускаем Settings**

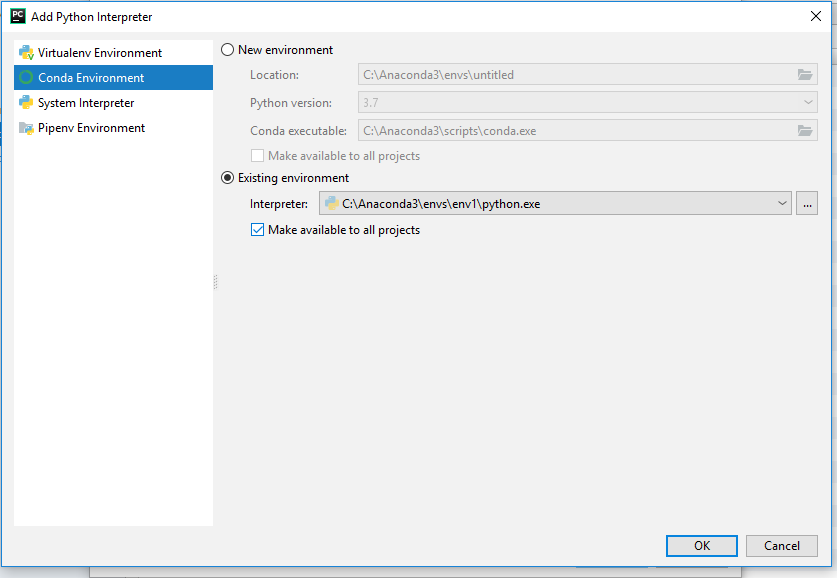


Далее Project Interpretator – Show All – Добавить (+)

(Подробно показано в видео из урока)

Далее слева выбираем Conda Environment. Выбираем Existing environment (Существующее окружение, ведь мы его уже создали), нажимаем справа на три точки и находим файл python.exe в папке нашего созданного виртуального окружения env1, которое находится в папке с Анакондой. В моем случае это C:\Anaconda3\envs\env1\python.exe

Нажимаем ОК и ждем результат.



После некоторого ожидания мы получаем одно виртуальное окружение с Анакондой и можем использовать все установленные там библиотеки.

PS. При подключении в PyCharm библиотеки pillow может возникнуть ошибка (ImportError: DLL load failed).

Это происходит из-за несовместимости версий pillow и python.

Проблема решается установкой более старой версии pillow.

**2.1 Типы данных. Переменные**

В реальной жизни мы совершаем различные действия над окружающими нас предметами, или объектами. Мы меняем их свойства, наделяем новыми функциями. По аналогии с этим компьютерные программы также управляют объектами, только виртуальными, цифровыми.

Очевидно, данные бывают разными. Часто компьютерной программе приходится работать с числами и строками. Так на прошлом уроке мы работали с числами, выполняя над ними арифметические операции. Операция сложения выполняла изменение первого числа на величину второго, а умножение увеличивало одно число в количество раз, соответствующее второму.

Числа в свою очередь также бывают разными: целыми, вещественными, могут иметь огромное значение или очень длинную дробную часть.

При знакомстве с языком программирования Python мы столкнемся с тремя типами данных:

целые числа (тип int) – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0).

числа с плавающей точкой (тип float) – дробные, они же вещественные, числа (например, 1.45, -3.789654, 0.00453). Примечание: для разделения целой и дробной частей здесь используется точка, а не запятая.

строки (тип str) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; одиночный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.

**2.2 Операции в программировании**

Операция – это выполнение каких-либо действий над данными, которые в данном случае именуют операндами. Само действие выполняет оператор – специальный инструмент. Если бы вы выполняли операцию постройки стола, то вашими операндами были бы доска и гвоздь, а оператором – молоток.

Оператор и операнды

Так в математике и программировании символ плюса является оператором операции сложения по отношению к числам. В случае строк этот же оператор выполняет операцию конкатенации, то есть соединения.

**>>> 10.25 + 98.36**

**108.61**

**>>> 'Hello' + 'World'**

**'HelloWorld'**

Здесь следует для себя отметить, что то, что делает оператор в операции, зависит не только от него, но и от типов данных, которыми он оперирует. Молоток в случае нападения на вас крокодила перестанет играть роль строительного инструмента. Однако в большинстве случаев операторы не универсальны. Например, знак плюса неприменим, если операндами являются, с одной стороны, число, а с другой – строка.

**>>> 1 + 'a'**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: unsupported operand type(s) for +:**

**'int' and 'str'**

Здесь в строке TypeError: unsupported operand type(s) for +: 'int' and 'str' интерпретатор сообщает, что произошла ошибка типа – неподдерживаемый операнд для типов int и str.

**2.3 Изменение типов данных**

Приведенную выше операцию все-таки можно выполнить, если превратить число 1 в строку "1". Для изменения одних типов данных в другие в языке Python предусмотрен ряд встроенных в него функций (что такое функция в принципе, вы узнаете в других уроках). Поскольку мы пока работаем только с тремя типами (int, float и str), рассмотрим вызовы соответствующих им функций – int(), float(), str().

**>>> str(1) + 'a'**

**'1a'**

**>>> int('3') + 4**

**7**

**>>> float('3.2') + int('2')**

**5.2**

**>>> str(4) + str(1.2)**

**'41.2'**

Эти функции преобразуют то, что помещается в их скобки соответственно в целое число, вещественное число или строку. Однако преобразовать можно не все:

**>>> int('hi')**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**ValueError: invalid literal for int() with**

**base 10: 'hi'**

Здесь возникла ошибка значения (ValueError), так как передан литерал (в данном случае строка с буквенными символами), который нельзя преобразовать к числу с основанием 10. Однако функция int не такая простая:

**>>> int('101', 2)**

**5**

**>>> int('F', 16)**

**15**

Если вы знаете о различных системах счисления, то поймете, что здесь произошло.

Обратим внимание еще на одно. Данные могут называться значениями, а также литералами. Эти три понятия ("данные", "значение", "литерал") не обозначают одно и то же, но близки и нередко употребляются как синонимы. Чтобы понять различие между ними, места их употребления, надо изучить программирование глубже.

**2.4 Переменные**

Данные хранятся в ячейках памяти компьютера. Когда мы вводим число, оно помещается в какую-то ячейку памяти. Но как потом узнать, куда именно? Как впоследствии обращаться к этим данными? Нужно как-то запомнить, пометить соответствующую ячейку.

Раньше, при написании программ на машинном языке, обращение к ячейкам памяти осуществляли с помощью указания их регистров, то есть конкретно сообщали, куда положить данные и откуда их взять. Однако с появлением ассемблеров при обращении к данным стали использовать словесные переменные, что куда удобней для человека.

Механизм связи между переменными и данными может различаться в зависимости от языка программирования и типов данных. Пока достаточно запомнить, что в программе данные связываются с каким-либо именем и в дальнейшем обращение к ним возможно по этому имени-переменной.

Слово "переменная" обозначает, что сущность может меняться, она непостоянна. Действительно, вы увидите это в дальнейшем, одна и та же переменная может быть связана сначала с одними данными, а потом – с другими. То есть ее значение может меняться, она переменчива.

В программе на языке Python, как и на большинстве других языков, связь между данными и переменными устанавливается с помощью знака =. Такая операция называется присваивание (также говорят "присвоение"). Например, выражение sq = 4 означает, что на объект, представляющий собой число 4, находящееся в определенной области памяти, теперь ссылается переменная sq, и обращаться к этому объекту следует по имени sq.

Имена переменных могут быть любыми. Однако есть несколько общих правил их написания:

Желательно давать переменным осмысленные имена, говорящие о назначении данных, на которые они ссылаются.

Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).

Имя переменной должно начинаться с буквы или символа подчеркивания (\_), но не с цифры.

Имя переменной не должно содержать пробелы.

Чтобы узнать значение, на которое ссылается переменная, находясь в режиме интерпретатора, достаточно ее вызвать, то есть написать имя и нажать Enter.

**>>> sq = 4**

**>>> sq**

**4**

Вот более сложный пример работы с переменными в интерактивном режиме:

**>>> apples = 100**

**>>> eat\_day = 5**

**>>> day = 7**

**>>> apples = apples - eat\_day \* day**

**>>> apples**

**65**

Здесь фигурируют три переменные: apples, eat\_day и day. Каждой из них присваивается свое значение. Выражение apples = apples - eat\_day \* day сложное. Сначала выполняется подвыражение, стоящее справа от знака равенства. После этого его результат присваивается переменной apples, в результате чего ее старое значение (100) теряется. В подвыражении apples - eat\_day \* day вместо имен переменных на самом деле используются их значения, то есть числа 100, 5 и 7.

**3.1 Ввод и вывод данных**

Мы уже встречались с функцией print(). Она отвечает за вывод данных, по умолчанию на экран. Если код содержится в файле, то без нее не обойтись. В интерактивном режиме в ряде случаев можно обойтись без нее.

Ввод данных в программу и их вывод важны в программировании. Без ввода программы делали бы одно и то же, исключая случаи, когда в них самих генерируются случайные значения. Вывод позволяет увидеть, использовать, передать дальше результат работы программы.

Обычно требуется, чтобы программа обрабатывала какой-то диапазон различных входных данных, которые поступают в нее из внешних источников. В качестве последних могут выступать файлы, клавиатура, сеть, выходные данные из другой программы. В свою очередь вывод данных, например, возможен в файл, по сети, в базу данных, на принтер. Однако нередко информацию просто выводят на экран монитора.

Можно сказать, что программа – это открытая система, которая обменивается чем-либо с внешней для нее средой. Если живой организм в основном обменивается веществом и энергией, то программа – данными, информацией.

**3.2 Вывод данных. Функция print()**

Что такое функция в программировании, узнаем позже. Пока будем считать, что print() – это такая команда языка Python, которая выводит то, что в ее скобках на экран.

**>>> print(1032)**

**1032**

**>>> print(2.34)**

**2.34**

**>>> print("Hello")**

**Hello**

В скобках могут быть любые типы данных. Кроме того, количество данных может быть различным:

**>>> print("a:", 1)**

**a: 1**

**>>> one = 1**

**>>> two = 2**

**>>> three = 3**

**>>> print(one, two, three)**

**1 2 3**

Можно передавать в функцию print() как непосредственно литералы (в данном случае "a:" и 1), так и переменные, вместо которых будут выведены их значения. Аргументы функции (то, что в скобках), разделяются между собой запятыми. В выводе вместо запятых значения разделены пробелом.

Если в скобках стоит выражение, то сначала оно выполняется, после чего print() уже выводит результат данного выражения:

**>>> print("hello" + " " + "world")**

**hello world**

**>>> print(10 - 2.5/2)**

**8.75**

В print() предусмотрены дополнительные параметры. Например, через параметр sep можно указать отличный от пробела разделитель строк:

**>>> print("Mon", "Tue", "Wed", "Thu",**

**... "Fri", "Sat", "Sun", sep="-")**

**Mon-Tue-Wed-Thu-Fri-Sat-Sun**

**>>> print(1, 2, 3, sep="//")**

**1//2//3**

Параметр end позволяет указывать, что делать, после вывода строки. По умолчанию происходит переход на новую строку. Однако это действие можно отменить, указав любой другой символ или строку:

**>>> print(10, end="")**

**10**

**>>>**

Обычно end используется не в интерактивном режиме, а в скриптах, когда несколько выводов подряд надо разделить не переходом на новую строку, а, скажем, запятыми. Сам переход на новую строку обозначается символом '\n'. Если присвоить это значение параметру end, то никаких изменений в работе функции print вы не увидите, так как это значение и так присвоено по умолчанию:

**>>> print(10, end='\n')**

**10**

**>>>**

Однако, если надо отступить на одну дополнительную строку после вывода, то можно сделать так:

**>>> print(10, end='\n\n')**

**10**

**>>>**

В функцию print нередко передаются так называемые форматированные строки, хотя по смыслу их правильнее называть строки-шаблоны. Никакого отношения к самому print они не имеют. Когда такая строка находится в скобках print(), интерпретатор сначала согласно заданному в ней формату преобразует ее к обычной строке, после чего передает результат в print().

Форматирование может выполняться в так называемом старом стиле или с помощью строкового метода format. Старый стиль также называют Си-стилем, так как он схож с тем, как происходит вывод на экран в языке C. Рассмотрим пример:

**>>> pupil = "Ben"**

**>>> old = 16**

**>>> grade = 9.2**

**>>> print("It's %s, %d. Level: %f" %**

**... (pupil, old, grade))**

**It's Ben, 16. Level: 9.200000**

Здесь вместо трех комбинаций символов %s, %d, %f подставляются значения переменных pupil, old, grade. Буквы s, d, f обозначают типы данных – строку, целое число, вещественное число. Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание %s.

Хотя в качестве значения переменной grade было указано число 9.2, на экран оно вывелось с дополнительными нулями. Чтобы указать, сколько требуется знаков после запятой, надо перед f поставить точку, после нее указать желаемое количество знаков в дробной части:

**>>> print("It's %s, %d. Level: %.1f"**

**... % (pupil, old, grade))**

**It's Ben, 16. Level: 9.2**

Теперь посмотрим на метод format():

**>>> print("This is a {0}. It's {1}."**

**... .format("ball", "red"))**

**This is a ball. It's red.**

**>>> print("This is a {0}. It's {1}."**

**... .format("cat", "white"))**

**This is a cat. It's white.**

**>>> print("This is a {0}. It's {1} {2}."**

**... .format(1, "a", "number"))**

**This is a 1. It's a number.**

В строке в фигурных скобках указаны номера данных, которые будут сюда подставлены. Далее к строке применяется метод format(). В его скобках указываются сами данные (можно использовать переменные). На нулевое место подставится первый аргумент метода format(), на место с номером 1 – второй и т. д.

На самом деле возможности метода format() существенно шире, и для их изучения понадобился бы отдельный урок. Нам пока будет достаточно этого.

В новых релизах Питона появился третий способ создания форматированных строк – f-строки. Перед их открывающей кавычкой прописывается буква f. В самой строке внутри фигурных скобок записываются выражения на Python, которые исполняются, когда интерпретатор преобразует строку-шаблон в обычную.

**>>> a = 10**

**>>> b = 1.33**

**>>> c = 'Box'**

**>>> print(f'qty - {a:5}, goods - {c}')**

**qty - 10, goods - Box**

**>>> print(f'price - {b + 0.2:.1f}')**

**price - 1.5**

В примере число 5 после переменной a обозначает количество знакомест, отводимых под вывод значения переменной. В выражении b + 0.2:.1f сначала выполняется сложение, после этого значение округляется до одного знака после запятой.

**3.3 Ввод данных. Функция input()**

За ввод в программу данных с клавиатуры в Python отвечает функция input. Когда вызывается эта функция, программа останавливает свое выполнение и ждет, когда пользователь введет текст. После этого, когда он нажмет Enter, функция input() заберет введенный текст и передаст его программе, которая уже будет обрабатывать его согласно своим алгоритмам.

Если в интерактивном режиме ввести команду input(), то ничего интересного вы не увидите. Компьютер будет ждать, когда вы что-нибудь введете и нажмете Enter или просто нажмете Enter. Если вы что-то ввели, это сразу же отобразиться на экране:

**>>> input()**

**Yes!**

**'Yes!'**

Функция input() передает введенные данные в программу. Их можно присвоить переменной. В этом случае интерпретатор не выводит строку сразу же:

**>>> answer = input()**

**No, it is not.**

В данном случае строка сохраняется в переменной answer, и при желании мы можем вывести ее значение на экран:

**>>> answer**

**'No, it is not.'**

При использовании функции print() кавычки в выводе опускаются:

**>>> print(answer)**

**No, it is not.**