

ОСНОВНЫЕ ТИПЫ ДАННЫХ PYTHON

УСТАНОВКА PYTHON И PYCHARM

ОСНОВНЫЕ ТИПЫ ДАННЫХ PYTHON

ПРАВИЛА ОФОРМЛЕНИЯ КОДА

Гаврилов Денис Андреевич, преподаватель кафедры СИ ФИТ НГУ

Установка Python в Windows.
Установка PyCharm в Windows.

Репозиторий пакетов
программных средств PyPI.
Менеджер пакетов pip.

Установка пакетов в Python с
использованием pip.

Первая программа в среде
интерпретатора Python.

УСТАНОВКА PYTHON И PYCHARM

УСТАНОВКА PYTHON В WINDOWS

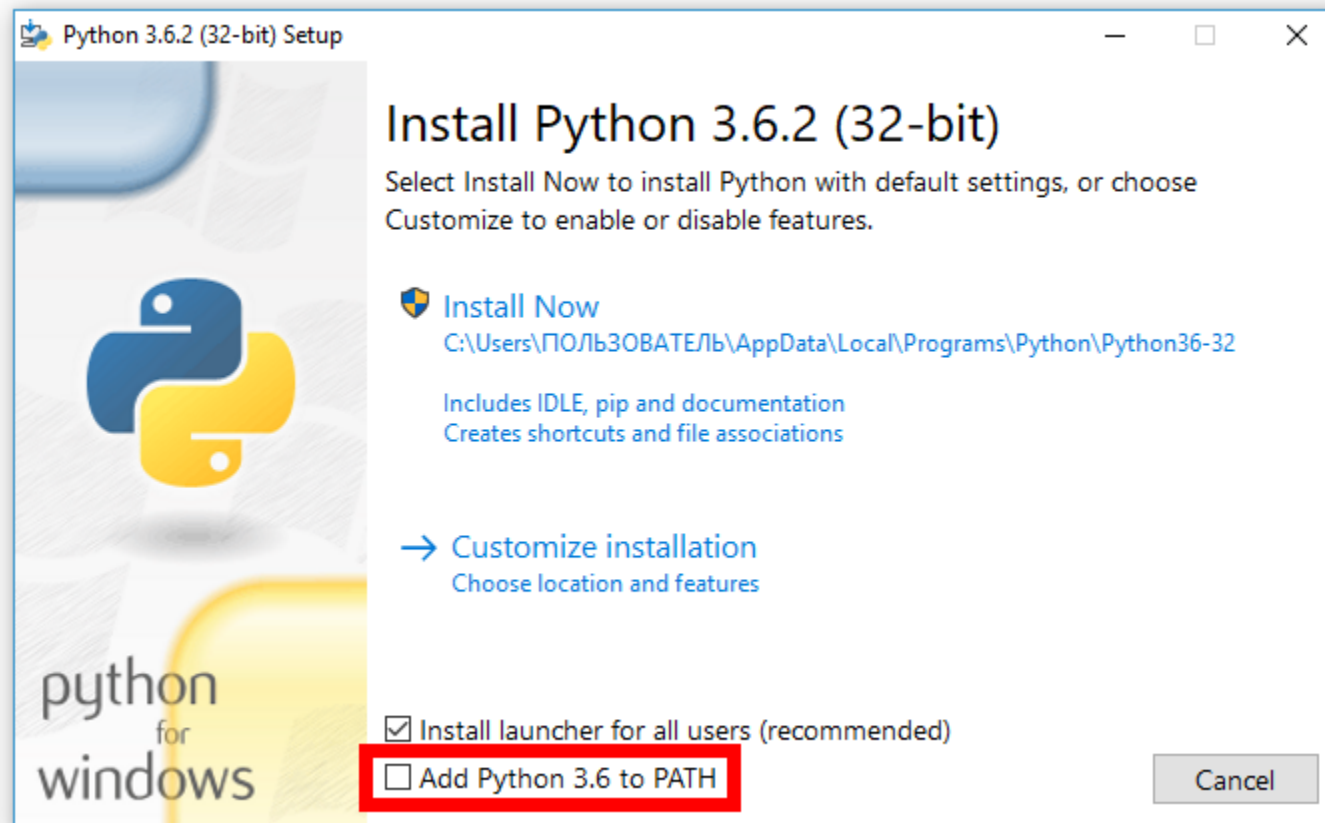
Шаг 1. Скачайте дистрибутив Python с официального сайта: [Download Python | Python.org](https://www.python.org/downloads/)

Шаг 2. Запустите скачанный установочный файл. Выберите способ установки:

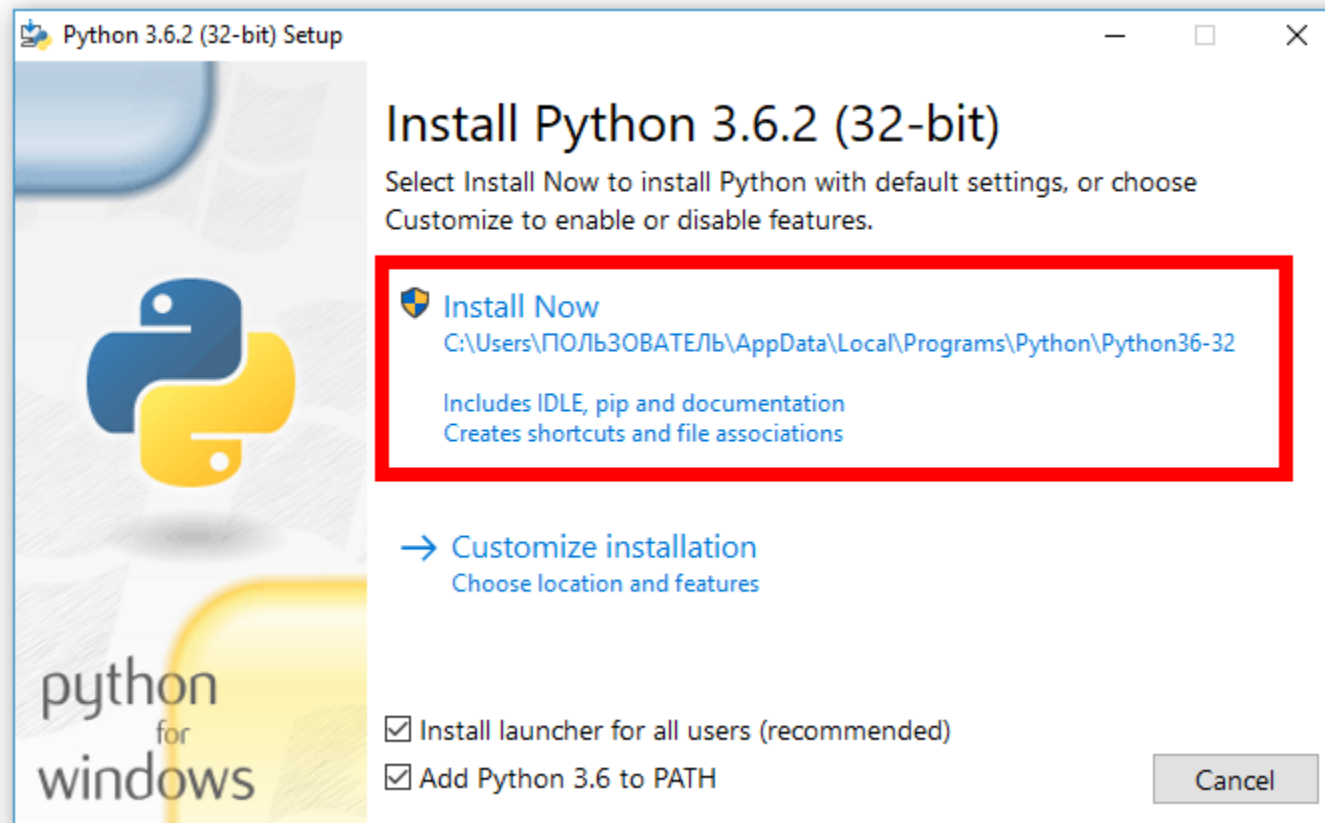
Install Now – в папку по указанному в окне адресу будут установлены интерпретатор, IDLE (интегрированная среда разработки), пакетный менеджер (pip), документация, установлены связи файлов .py с интерпретатором Python.

Customize installation – настраиваемая установка. Рекомендуется отметить, как минимум, опции *Documentation, pip, tcl/tk and IDLE*.

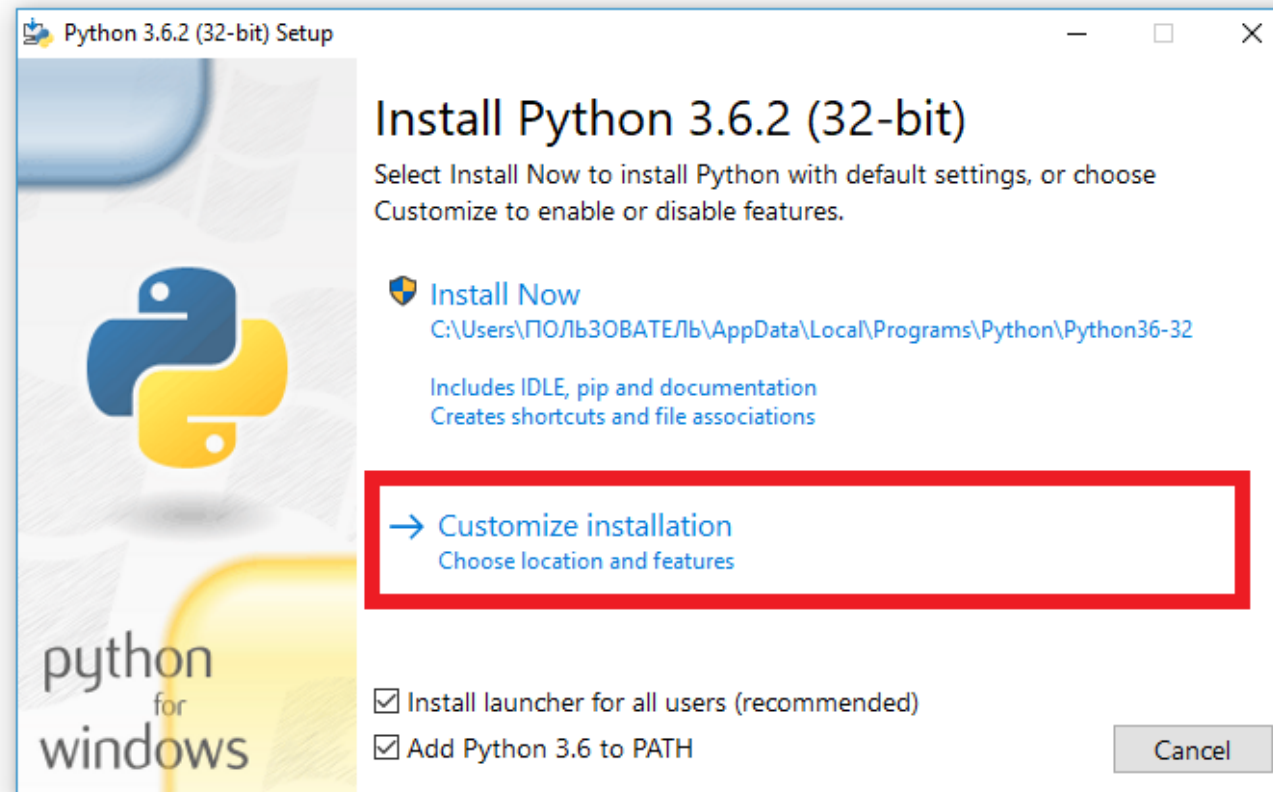
УСТАНОВКА PYTHON В WINDOWS



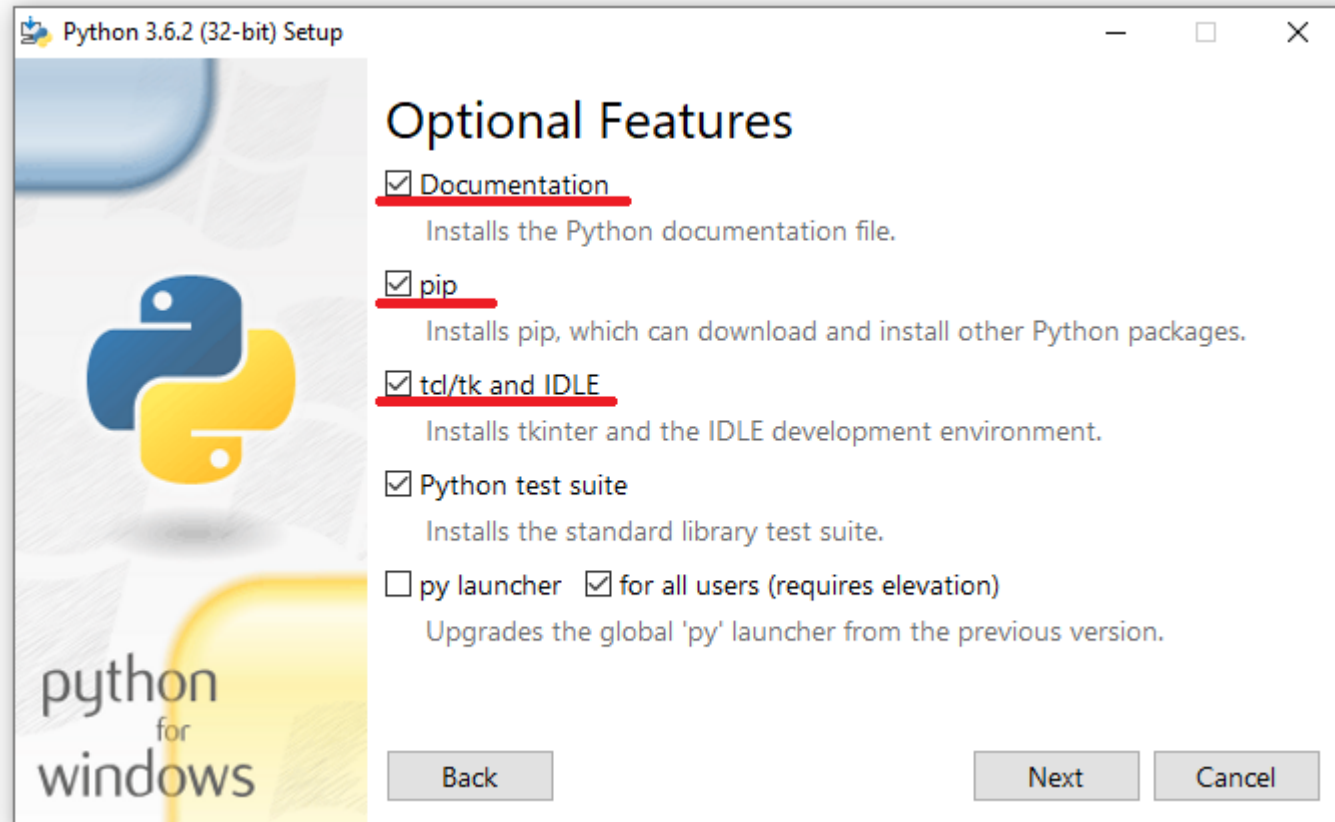
УСТАНОВКА PYTHON В WINDOWS



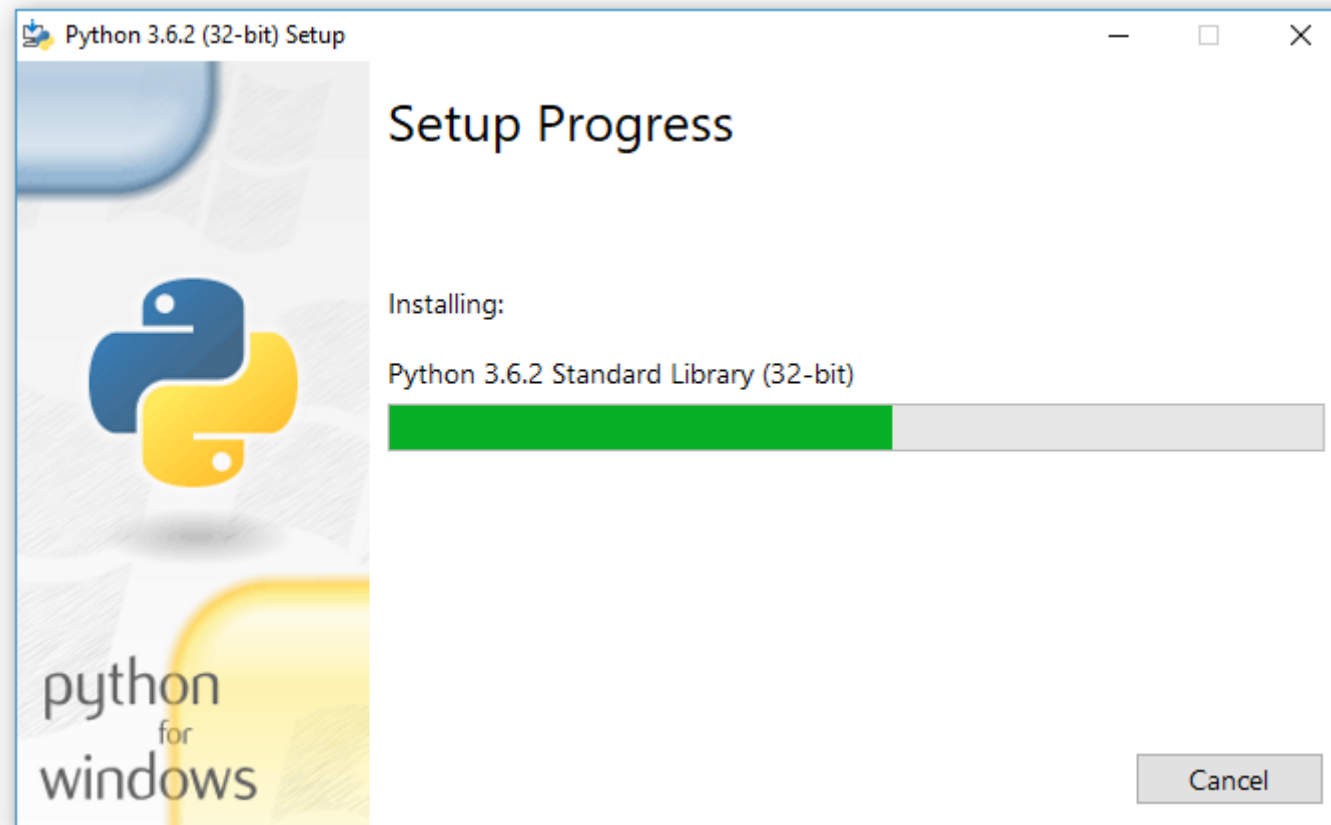
УСТАНОВКА PYTHON В WINDOWS



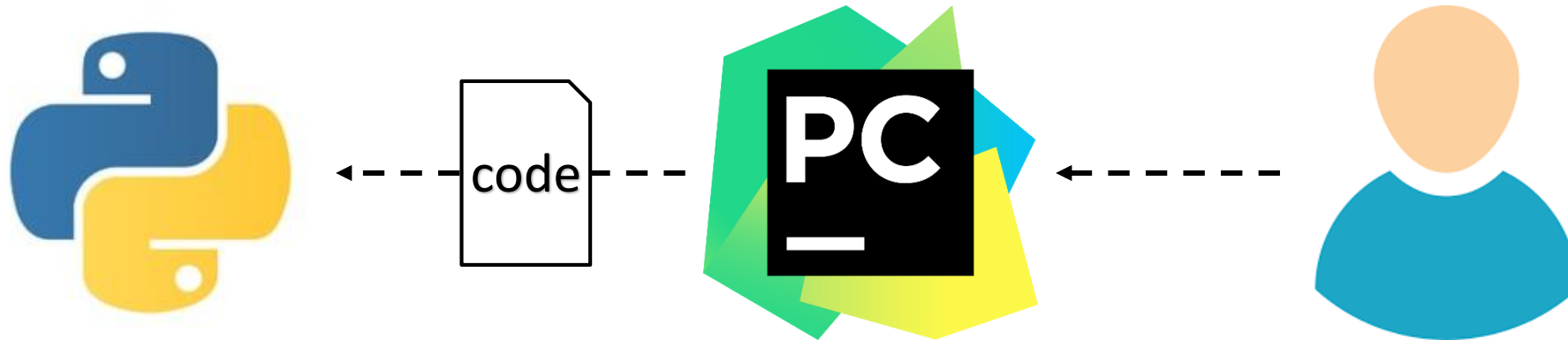
УСТАНОВКА PYTHON В WINDOWS



УСТАНОВКА PYTHON В WINDOWS



УСТАНОВКА PYCHARM В WINDOWS



Программный код Python можно писать даже в Блокноте, но это не очень удобно. Гораздо лучше – и проще – воспользоваться средой, которая (среди прочего) будет подсвечивать набираемый код и автоматически указывать на различные ошибки в нем.

УСТАНОВКА PYCHARM В WINDOWS

Для установки выполните следующее:

Шаг 1. Скачайте дистрибутив по ссылке: [PyCharm: Python IDE для профессиональных разработчиков от JetBrains](#)

Выбираем бесплатный вариант (*Community*);

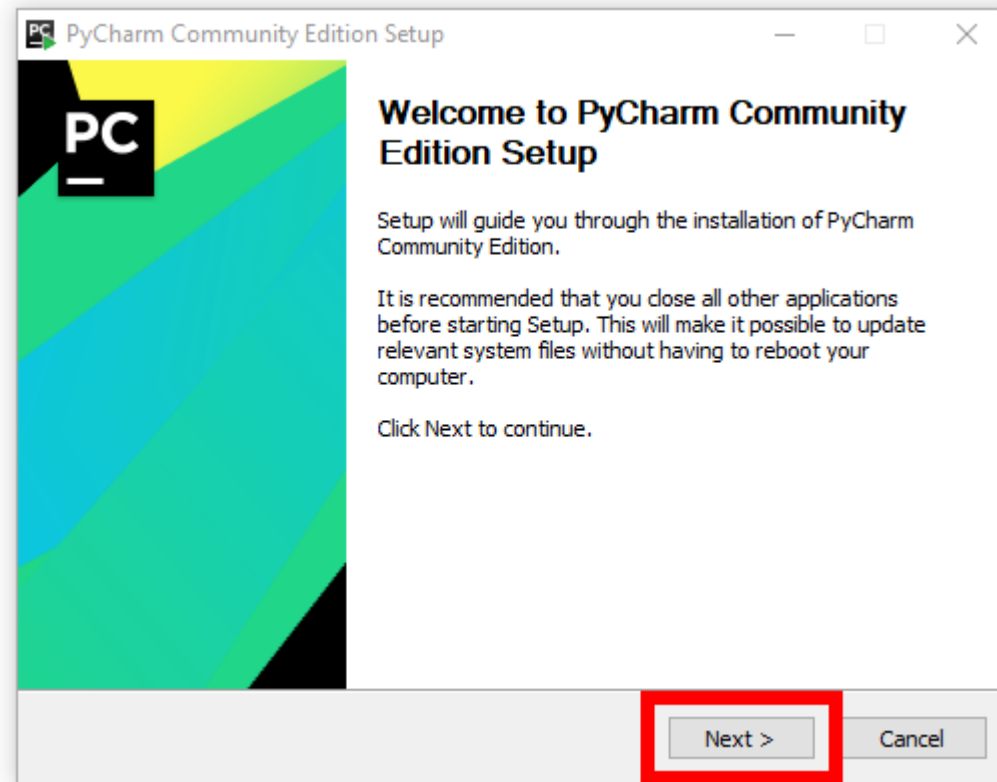
Шаг 2. Запустите дистрибутив PyCharm на выполнение.

Выберите путь установки программы (*Destination Folder*);

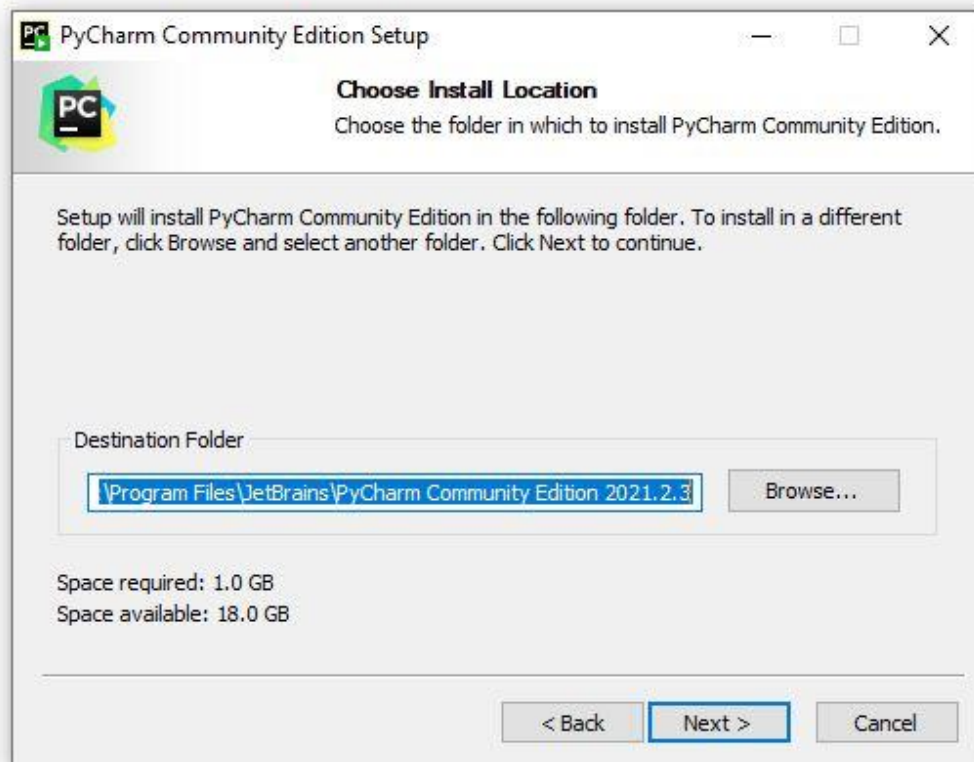
Укажите ярлыки для создания на рабочем столе;

Включите опцию *Create associations (.py)* для ассоциации файлов Python с PyCharm.

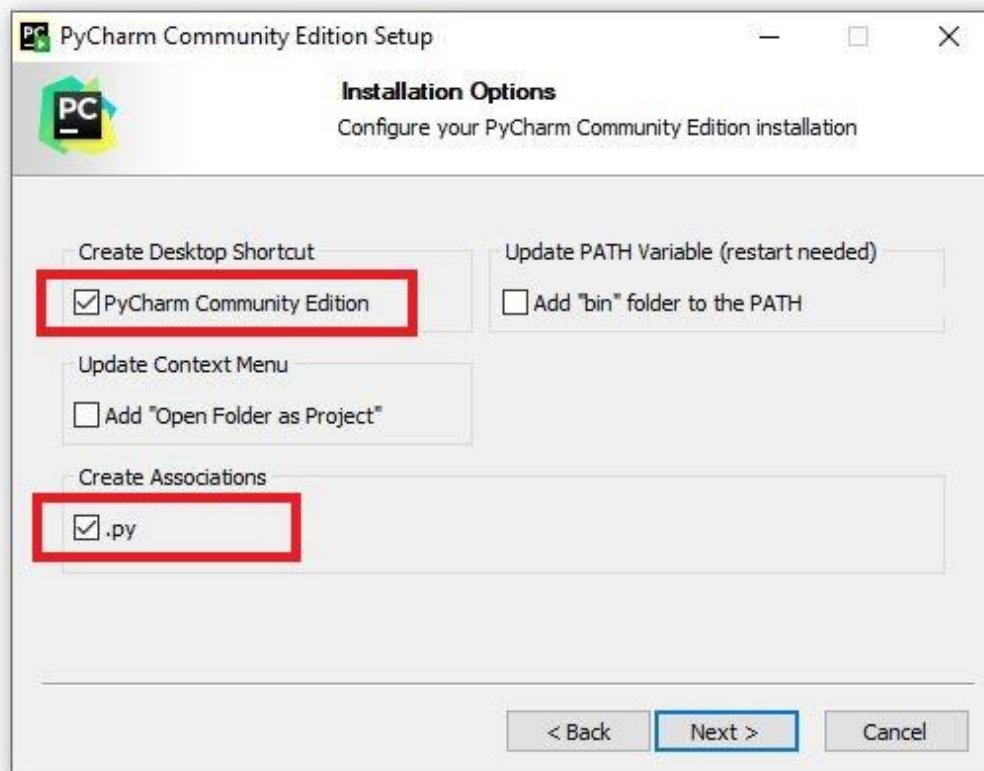
УСТАНОВКА PYCHARM В WINDOWS



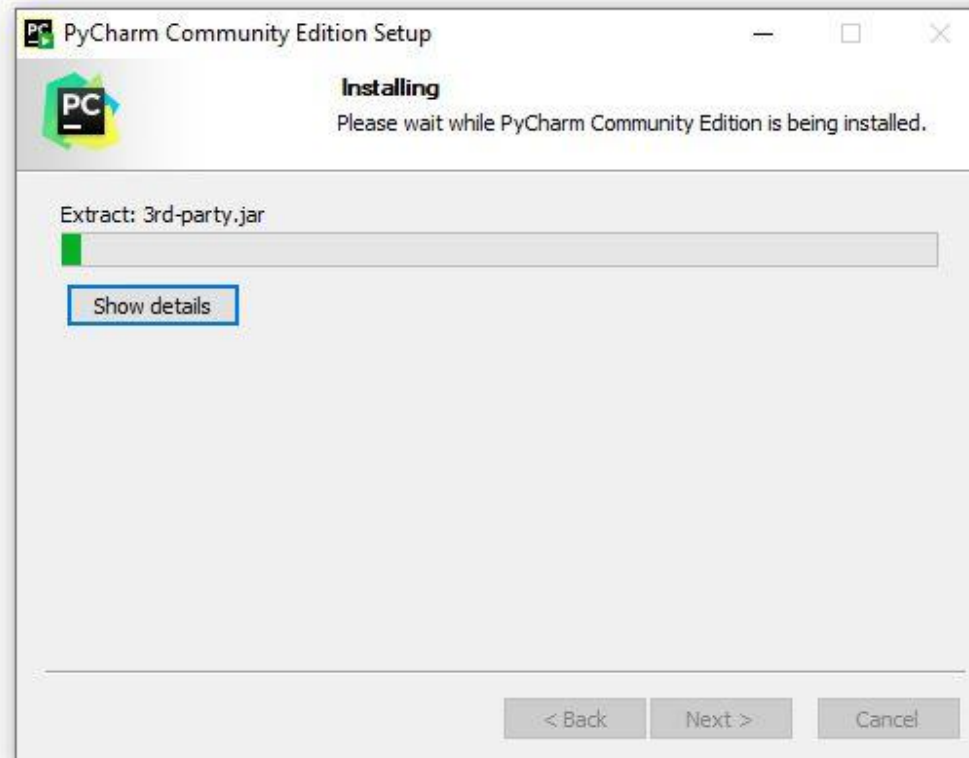
УСТАНОВКА PYCHARM В WINDOWS



УСТАНОВКА PYCHARM В WINDOWS



УСТАНОВКА PYCHARM В WINDOWS



РЕПОЗИТОРИЙ ПАКЕТОВ ПРОГРАММНЫХ СРЕДСТВ PYPI

PyPI (Python Package Index) – открытый репозиторий (хранилище), в котором можно найти пакеты для решения различных задач. Необходимые пакеты можно просто подключить к своему проекту и активировать их функционал.

Для скачивания и установки нужных пакетов используется **утилита командной строки pip**.

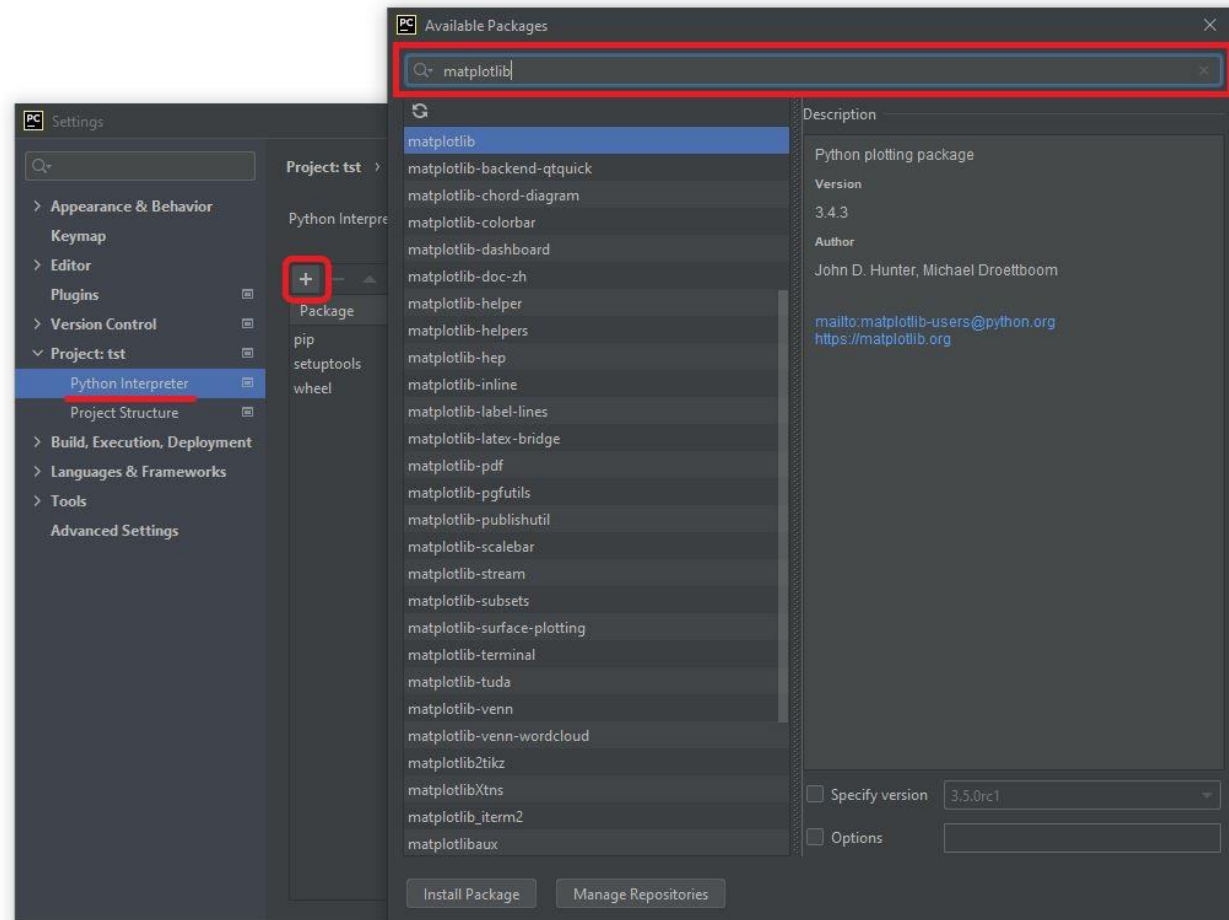
Начиная с версии Python 2.7.9, **pip** устанавливается автоматически.

РЕПОЗИТОРИЙ ПАКЕТОВ ПРОГРАММНЫХ СРЕДСТВ РYPI

Начиная с версии Python 2.7.9, `pip` устанавливается автоматически.

- Чтобы проверить наличие модуля **pip**, нужно войти в меню *File/Settings/Project Interpreter* и найти модуль **pip** в списке.
- Если модуль **pip** отсутствует, то его можно загрузить здесь же по кнопке *Install Package* внизу экрана. Так же, как и другие пакеты, например, Plotly или Matplotlib.

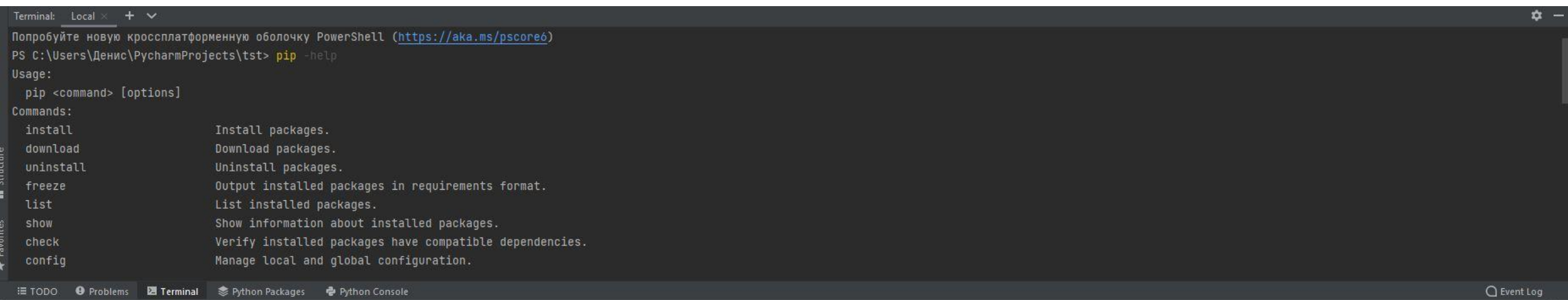
РЕПОЗИТОРИЙ ПАКЕТОВ ПРОГРАММНЫХ СРЕДСТВ РYPI



РЕПОЗИТОРИЙ ПАКЕТОВ ПРОГРАММНЫХ СРЕДСТВ РYPI

После скачивания и установки, утилиту **pip** можно запускать или через терминал Windows, или в терминальном окне PyCharm:

>pip <аргументы>



```
Terminal: Local x + v
Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)
PS C:\Users\Денис\PycharmProjects\tst> pip -help
Usage:
  pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  check             Verify installed packages have compatible dependencies.
  config            Manage local and global configuration.
```

РЕПОЗИТОРИЙ ПАКЕТОВ ПРОГРАММНЫХ СРЕДСТВ РYPI

После скачивания и установки, утилиту **pip** можно запускать или через терминал Windows, или в терминальном окне PyCharm:

```
>pip <аргументы>
```

Также **pip** можно запустить через интерпретатор Python:

```
>Python -m pip <аргументы>
```

ПЕРВАЯ ПРОГРАММА В СРЕДЕ PYCHARM

Для создания простейшего приложения в среде PyCharm нужно выполнить следующее:

- Создать новый проект (Выбрать меню *File/New Project*, в строке *Location* задать имя проекта, например, **MyProject**, нажать кнопку *Create*).
- Создать файл, в котором будет содержаться программный код Python (в строке **MyProject** по правой кнопке мыши выбрать *New/Python File*, в открывшемся окне выбрать опцию *Python File* и задать имя программы, например, **MyFirstProgram**).
- В открывшемся окне редактора набрать код; в качестве примера: `print("Hello, world!")`
- По правой кнопке мыши выбрать *Run/ MyFirstProgram* и запустить программу на выполнение.

Типы данных Python.

Переменные и константы.

Простые типы данных.

Числа, строки, списки.

Условный оператор «if».

Операторы циклов «for» и
«while».

ОСНОВНЫЕ ТИПЫ ДАННЫХ PYTHON

ПЕРЕМЕННЫЕ И КОНСТАНТЫ

Для работы с данными используются **переменные**.

Переменная в Python указывает на область памяти компьютера, в которой хранятся данные.

Во многих языках программирования, прежде чем использовать переменные, их необходимо объявить, то есть указать, какого типа данные будут в них храниться.

В языке Python такой необходимости нет.

Тип данных переменной **определяется автоматически**, в момент присвоения переменной конкретных данных.

ПЕРЕМЕННЫЕ И КОНСТАНТЫ

Важно: В отличие от некоторых других языков, в Python **нет возможности объявить неизменяемую переменную**.

Для обозначения переменных, значения которых не должны меняться, существует договорённость именовать их прописными буквами.

```
A = 1  
b = 1  
b = b + A  
hello_string = "Hello, world!"  
print(hello_string, b)
```

ТИПЫ ДАННЫХ PYTHON

Тип данных характеризует одновременно:

- множество допустимых значений, которые могут принимать данные, принадлежащие к этому типу;
- набор операций, которые можно осуществлять над данными, принадлежащими к этому типу.

ТИПЫ ДАННЫХ PYTHON

Python поддерживает следующие (встроенные) типы данных:

- **Числовые типы данных:** int, long, float, complex;
- **Строковый тип данных:** str;
- **Последовательности:** list, tuple, range (список, кортеж, диапазон);
- **Двоичные типы данных:** bytes, bytearray, memoryview;
- **Тип данных словарь:** dict;
- **Логические типы данных:** bool; set, frozenset.

ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Числовой тип данных Python используется для хранения числовых значений.

int – содержит целые числа со знаком; *long* – неограниченной длины;

float – содержит числа с плавающей точкой с точностью до 15 десятичных знаков;

комплексный – содержит комплексные числа.

#int - целые числа

```
x=2147483647
```

```
print(x)
```

```
print(type(x))
```

float - вещественные числа

```
y=5.1
```

```
print(y)
```

```
print(type(y))
```

complex - комплексные числа

```
z=2+2i
```

```
print(z)
```

```
print(type(z))
```

ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Важно: если в результате операции получается значение, выходящее за рамки допустимого, тип *int* может быть неявно преобразован в *long*.

Значения типа *int* должны покрывать диапазон от -2147483648 до 2147483647, а точность целых произвольной точности (*long*) зависит от объема доступной памяти.

ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Набор операций над числами - достаточно стандартный как по семантике, так и по обозначениям:

```
print(1 + 1, 3 - 2, 2*2, 7/4, 5%3) # арифметические операции
print(3 < 4 < 6, 3 >= 5, 4 == 4, 4 != 4) # сравнения
print(1 << 8, 4 >> 2, ~4) # побитовые сдвиги и инверсия

for i, j in (0, 0), (0, 1), (1, 0), (1, 1):
    print(i, j, ":", i & j, i | j, i ^ j) # побитовые операции
```

СТРОКОВЫЕ ТИПЫ ДАННЫХ

В Python строки бывают двух типов: обычные и Unicode-строки.

Строки-константы можно задать в программе с помощью строковых литералов. Для литералов используются как апострофы ('), так и обычные двойные кавычки (").

Для многострочных литералов можно использовать утроенные апострофы или утроенные кавычки.

Управляющие последовательности (например, символ переноса строки) внутри строковых литералов задаются обратной косой чертой (\).

СТРОКОВЫЕ ТИПЫ ДАННЫХ

Примеры строк:

```
s1 = "строка1"  
s2 = 'строка2\nс переводом строки внутри'  
s3 = ""строка3  
с переводом строки внутри""  
u1 = u'\u043f\u0440\u0438\u0432\u0435\u0442' # привет  
u2 = u'Еще пример'
```

СТРОКОВЫЕ ТИПЫ ДАННЫХ

Набор операций над строками включает (в том числе) **конкатенацию** " + ", **повтор** " * ", **форматирование** " % ".

Полный набор методов (и их необязательных аргументов) можно получить в документации по Python.

```
"A" + "B" # 'AB'
```

```
"A"*10 # 'AAAAAAAAAAAA'
```

```
"%s %i" % ("abc", 12) # 'abc 12'
```

СПИСКИ

В "чистом" Python нет массивов с произвольным типом элемента.

Вместо них используются списки:

```
lst1 = [1, 2, 3]
```

```
lst2 = list("abcde")
```


СПИСКИ

Некоторые операции над списками:

Синтаксис	Семантика
$len(s)$	Длина s
$x \text{ in } s$	Проверка принадлежности элемента последовательности. В новых версиях Python можно проверять принадлежность подстроки строке. Возвращает True или False
$x \text{ not in } s$	То же, что и $\text{not } x \text{ in } s$
$s + s1$	Конкатенация
$s * n$ или $n * s$	Последовательность из n раз повторенной s . Если $n < 0$, возвращается пустая последовательность.
$s[i]$	Возвращает i -й элемент s или $len(s)+i$ -й, если $i < 0$
$s[i:j:d]$	Срез из последовательности s от i до j с шагом d (будет рассматриваться ниже)
$min(s)$	Наименьший элемент s
$max(s)$	Наибольший элемент s

СПИСКИ: СРЕЗЫ

В Python при взятии среза последовательности принято нумеровать не элементы, а промежутки между ними. Поначалу это кажется необычным, тем не менее, очень удобно для указания произвольных срезов.

Для записи срезов используется следующий синтаксис:

последовательность[нач:кон:шаг]

```
lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lst2 = lst[0:3] # [0, 1, 2]
```

```
lst3 = lst[0:10:3] # [0, 3, 6, 9]
```

```
lst4 = lst[-2:10:1] # [8, 9]
```

ОПЕРАТОРЫ ЦИКЛОВ «FOR» И «WHILE»

Если нужно, чтобы в программе какой-то блок кода повторился несколько раз (например, с целью обработки списка), используют операторы циклов.

Python поддерживает 2 вида циклов: цикл *while* и цикл *for*.

Обратите внимание, что **тело оператора** выделяется **отступом**.

```
lst = [1,2,3,4,5]
for i in lst:    # итератор i – элемент списка lst
    print(i)
```

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

ОПЕРАТОРЫ ЦИКЛОВ «FOR» И «WHILE»

Если нужно, чтобы в программе какой-то блок кода повторился несколько раз (например, с целью обработки списка), используют операторы циклов.

Python поддерживает 2 вида циклов: цикл *while* и цикл *for*.

Обратите внимание, что **тело оператора** выделяется **отступом**.

```
lst = [1,2,3,4,5]
for i in lst:    # итератор i – элемент списка lst
    print(i)
```

```
colors = ['red', 'green', 'blue', 'yellow']
for i in range(len(colors)):
    print(colors[i])
```

ОПЕРАТОРЫ ЦИКЛОВ «FOR» И «WHILE»

Важно: удалив часть кода с увеличением значения i , мы получим **бесконечный цикл**, а это уже **плохо**.

Бесконечные циклы являются **логическими ошибками**.

В случае необходимости в подобного рода циклах, для прерывания можно использовать встроенные средства Python, например, *break*:

```
i = 0
while i < 10:
    print(i)
```

```
i = 0
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1
```

ЛОГИЧЕСКИЙ ТИП ДАННЫХ

Подтип целочисленного типа для "канонического" обозначения логических величин.

Два значения: *True* (истина) и *False* (ложь) – все допустимые значения этого типа:

```
# bool - логический тип
z=True
z1=False
print(z,z1)
print(type(z))
print(type(z1))
```

УСЛОВНЫЙ ОПЕРАТОР «IF»

Инструкция *if-elif-else*, называемая **условным оператором** или **оператором ветвления** - основной инструмент для реализации алгоритма ветвления в Python.

Служит для выбора, какое действие следует выполнить в зависимости от значения переменных (или выражения) в момент проверки условия.

```
if number > 10:  
    print("число больше 10")  
elif number == 7:  
    print("число равно 7")  
else:  
    print("число меньше или равно 10 и не равно 7")
```

```
# Если число больше 10, то  
# напечатать: "число больше 10"  
# Или же если число больше 10, то  
# напечатать: "число равно 7"  
# В противном случае  
# напечатать: ("число меньше или равно 10 и не равно 7")
```

УСЛОВНЫЙ ОПЕРАТОР «IF»

Инструкция *if-elif-else*, называемая **условным оператором** или **оператором ветвления** - основной инструмент для реализации алгоритма ветвления в Python.

Служит для выбора, какое действие следует выполнить в зависимости от значения переменных (или выражения) в момент проверки условия.

```
if (number <= 10) and (number != 7):  
    print("число меньше или равно 10 и не равно 7")
```

```
# Если число меньше или равно 10 и не равно 7  
# напечатать: ("число меньше или равно 10 и не равно 7")
```


ИЗМЕНЯЕМЫЕ И НЕИЗМЕНЯЕМЫЕ ТИПЫ ДАННЫХ

Типы данных **списки** и **словари** являются изменяемыми. Это значит, что их элементы можно изменять.

Типы данных **числа**, **строки**, **кортежи** являются неизменяемыми.

```
lst=[1,2,3]
print(lst) # [1, 2, 3]

arr[0]=5
print(arr) # [5, 2, 3]
```

```
stroka='Елена'
print(stroka)
print(stroka[0]) # выводится 'Л'
#stroka[0]='Г'   # попытка изменить первый символ выдаст ошибку

stroka='вася'    # изменить значение целиком (переприсвоить) можно
print(stroka)
```

Отступы и пробелы в
выражениях и инструкциях.

Именованние констант,
переменных и функций.

Комментарии и строки
документации.

Запись циклов.

ПРАВИЛА ОФОРМЛЕНИЯ КОДА PYTHON (CODESTYLE)

ПРАВИЛА ОФОРМЛЕНИЯ КОДА

Это набор правил и соглашений, используемых группой разработчиков при написании исходного кода на некотором языке программирования.

Наличие общего стиля программирования:

- Облегчает понимание и поддержание исходного кода;
 - в т.ч. написанного более чем одним программистом;
- Упрощает взаимодействие нескольких человек при разработке программного обеспечения.

ПРАВИЛА ОФОРМЛЕНИЯ КОДА

Для языка Python разработан официальный стиль.

С оригинальным текстом "Python Style Guide" можно ознакомиться по адресу:

<http://www.python.org/doc/essays/styleguide>

ОТСТУПЫ И ПРОБЕЛЫ В ВЫРАЖЕНИЯХ И ИНСТРУКЦИЯХ

Рекомендуется использовать 4 пробела на каждый уровень отступа.

Важно: Python 3 запрещает смешивание табуляции и пробелов в отступах. Код, в котором используются и те, и другие типы отступов, должен быть исправлен так, чтобы отступы в нем были расставлены только с помощью пробелов (автоматизировано в IDLE).

```
if space_number == 4:  
    print("правильный отступ")  
elif tab_number > 0:  
    print("неправильный отступ")  
else:  
    print("отсутствующий отступ")
```

ИМЕНОВАНИЕ КОНСТАНТ, ПЕРЕМЕННЫХ И ФУНКЦИЙ

Старайтесь выбирать максимально информативные имена.

Избегайте односимвольных имен; исключая счетчики либо итераторы.

Никогда не используйте символы *l* (маленькая латинская буква «эль»), *O* (заглавная латинская буква «о») или *I* (заглавная латинская буква «ай») как однобуквенные идентификаторы. В некоторых шрифтах эти символы неотличимы от цифры один и нуля (*a* иногда и друг от друга).

```
good_name = 'Хорошее имя переменной'  
L = 'Допустимо, но лучше избегать'
```

```
l = 1  
I = 1  
O = 0
```

КОММЕНТАРИИ И СТРОКИ ДОКУМЕНТАЦИИ

- Комментарии должны точно отражать **актуальное** состояние кода.
- Рекомендуется писать комментарии на английском языке, если есть хотя бы небольшая вероятность того, что код будут читать специалисты, говорящие на языках, отличных от Вашего.
- Комментарии к фрагменту кода следует писать с тем же отступом, что и комментируемый код. После " # " должен идти одиночный пробел. Абзацы можно отделять строкой с " # " на том же уровне.
- **Хороший комментарий** не перефразирует программу, а **содержит дополнительную информацию** о действии программы **в терминах предметной области**.

КОММЕНТАРИИ И СТРОКИ ДОКУМЕНТАЦИИ

Пишите документацию для всех публичных модулей, функций, классов, методов. Строки документации необязательны для приватных методов, но лучше написать, что делает метод. Комментарий нужно писать после строки *c def*.

Очень важно, чтобы закрывающие кавычки стояли на отдельной строке. А еще лучше, если перед ними будет ещё и пустая строка.

```
def complex(real=0.0, imag=0.0):  
    """Form a complex number.  
  
    Keyword arguments:  
    real -- the real part (default 0.0)  
    imag -- the imaginary part (default 0.0)  
  
    """  
    if imag == 0.0 and real == 0.0:  
        return complex_zero  
    ...
```


ЗАПИСЬ ЦИКЛОВ

Проход по списку вперед:

```
colors = ['red', 'green', 'blue', 'yellow']  
for color in colors:  
    print(color)
```

```
colors = ['red', 'green', 'blue', 'yellow']  
for i in range(len(colors)):  
    print(colors[i])
```

Проход по списку назад:

```
colors = ['red', 'green', 'blue', 'yellow']  
for color in reversed(colors):  
    print(color)
```

```
colors = ['red', 'green', 'blue', 'yellow']  
for i in range(len(colors)-1, -1, -1):  
    print(colors[i])
```

БЛАГОДАРЮ ЗА ВНИМАНИЕ!

Гаврилов Денис Андреевич, преподаватель кафедры СИ ФИТ НГУ