

# Занятие #1

Введение в язык программирования Python.

Инфраструктура

# Разминка

Рассмотрим число  $17!$  (читается 17 факториал).

$n!$  - это число равное произведению чисел от 1 до  $n$ ,

т.е.  $1 * 2 * 3 * \dots * (n-2) * (n-1) * n$

На что заканчивается число  $17!$

Почему программисту задают такой вопрос на собеседовании?

Что ожидают от программиста? Что НЕ ожидают? Что делать, когда не знаешь решения? Не паниковать, исследовать задачу, рассуждать вслух.

Вопрос из собеседований Google:

На что заканчивается  $100!?$

На что заканчивается  $1000!?$

# Основные темы

- Что такое программирование
- Почему так много языков программирования
- Немного истории о языке программирования Python
- Установка инфраструктуры
- print, input
- if else
- Контроль версий. GitHub

# О себе

- Михаил Зимнев
- Разработчик, аналитик, руководитель проектов, руководитель подразделения в EPAM Systems.
- Некоторые из проектов:
  - Экспертная система диагностики в области абдоминальной хирургии
  - Заработная плата и тарификация медработников
  - Библиотечная система
  - Реестр акционеров, бэкофис брокерской компании
  - Поликом Про – система закупок и продаж, проекты «Выборжец», «Чупа-Чупс»
  - ФОРС СПб – внедрение IFS Applications (General DataComm, Franke, Ensto Electro, Нарвские электростанции)
  - EPAM Systems – внедрение SAP (Белгорхимпром, Hyundai Glovis Rus и др.)
  - и другие.

Что такое программирование. Почему язык программирования это язык? Программист – переводчик? С какого на какой?

1. Программирование = Алгоритмы + структуры данных (Н.Вирт)
2. Заказчик – пользователь (клиент всегда прав, всегда ли?)
3. Предметная область
4. Процесс разработки = анализ задачи, сбор данных, проектирование, кодирование, ОТЛАДКА, документирование, внедрение, сопровождение.

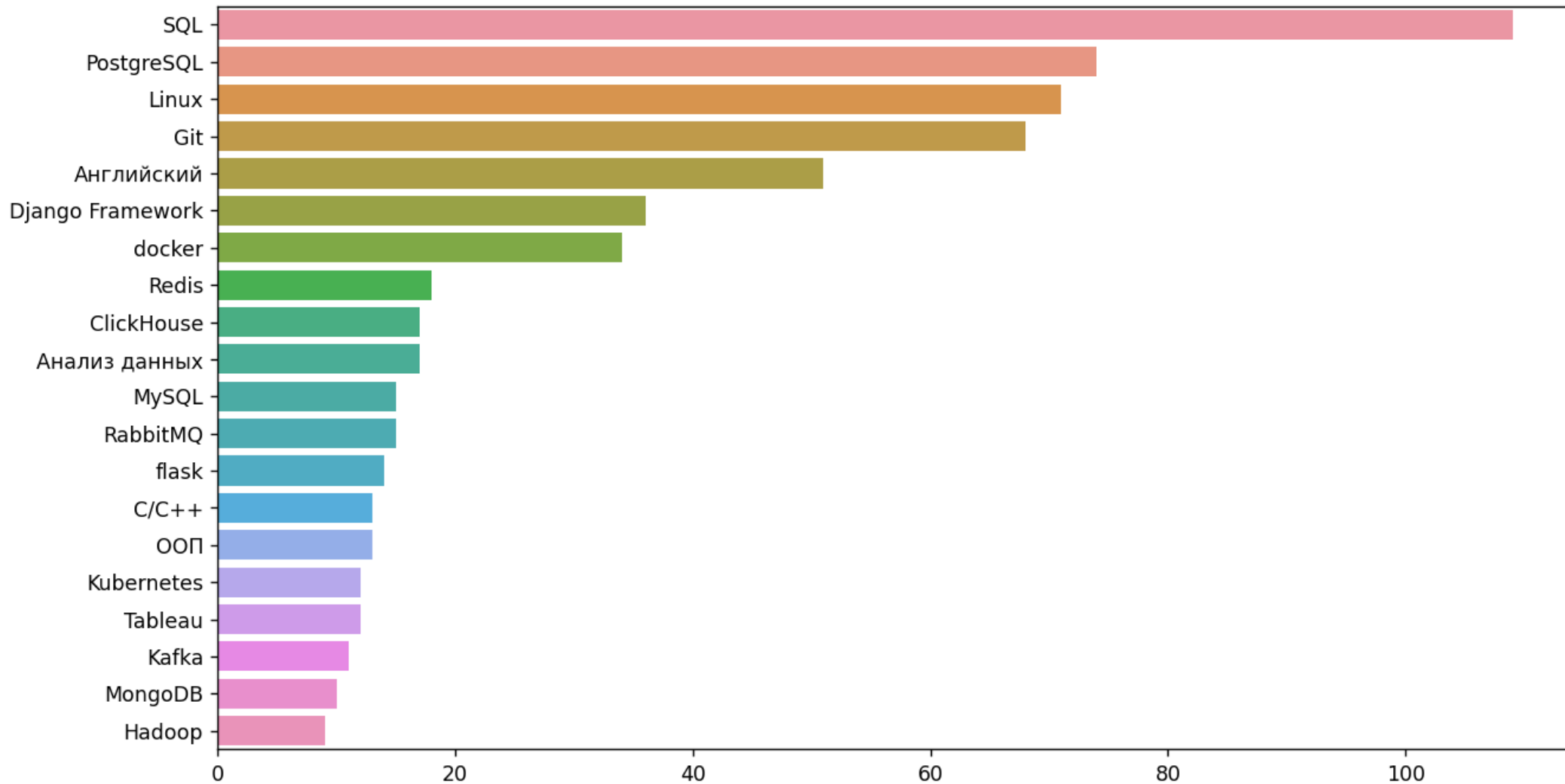
# Почему так много языков программирования? Какой выбрать?

1. Почему не ЕЯ (естественный язык)?
2. ТОР 11: [Python](#), [Javascript \(JS\)](#), [Java](#), [C/C++](#), [PHP](#), [Swift](#), [Golang \(Go\)](#), [C#](#), [Ruby](#), [Kotlin](#), [Perl](#)
3. Необходимые навыки: высокая самообучаемость, развитое аналитическое и абстрактное мышление.
4. Области применения: веб-разработка, десктопные графические интерфейсы, бизнес-приложения, машинное обучение (ML) и глубокое обучение (DL), наука о данных (Data science), искусственный интеллект (AI), игры, микроконтроллеры, анализ и визуализация данных.
5. Какие главные недостатки у Python?

# Как мы будем учиться программировать на языке Python?

- Учиться программировать + изучать язык Python
- Приобретать навык алгоритмического мышления
- Практика = Решение задач, домашние задания
- Вопросы на собеседованиях, тестовые задачи
- Задачники (codewars.com, projecteuler.net, hackerrank.com, и др.)
- Ресурсы (habr.com, pythonworld.ru и др.)
- Stepik.org – Поколение Python (для начинающих, продвинутых, профессионалов, ООП, SQL)
- Взаимопомощь (у тебя есть идея и у меня есть идея, теперь у нас по две идеи)
- Выпускная работа. Бизнес-задача, лучше если из собственной практики, анализ, выбор средств, решение. Презентация, код.
- Составление резюме для поиска работы разработчика на Python

# Какие ключевые навыки требуются в вакансиях, связанных с Python



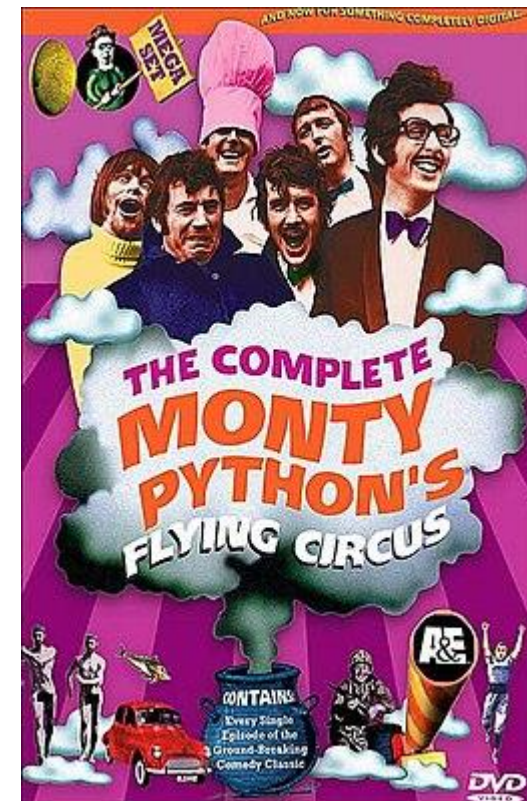


# Python – пакеты, модули, библиотеки

1. Web (Django, Flask, aiohttp, Requests, BeautifulSoup4 и др.)
2. Data Science (SciPy, NumPy, pandas, matplotlib, seaborn и др.)
3. Тестирование (PyTest, автоматизация с Selenium)
4. Машинное обучение (Tensorflow, PyTorch, Keras, и др.)
5. Системные утилиты (sys)
6. Графические приложения (PyQt, Tkinter и др.)
7. Мобильные приложения (Kivy)

# Python – немного истории

- 1. Автор - [Гвидо ван Россум](#)
- 2. Когда появился:
  - версия 0.9 – февраль 1991 г.
  - Python 1.0 — январь 1994 г.
  - Python 2.0 — 16 октября 2000 г.
  - Python 3.0 — 3 декабря 2008 г.
  - Python 3.12 – 2 октября 2023 г.
- 3. Название: в честь британского комедийного телешоу 1970-х «[Летающий цирк Монти Пайтона](#)»
- 4. Официальный сайт: [python.org](https://python.org)
- 5. Документация: <https://docs.python.org/3/>
- 6. На русском: <https://pydocs.ru/>,  
[https://digitology.tech/docs/python\\_3/index.html](https://digitology.tech/docs/python_3/index.html)



# Установка инфраструктуры

1. Установить (если не установлен) git для своей операционной системы:

<https://git-scm.com/download/win>

инструкция по установке: <https://nerdschalk.com/how-to-install-and-use-git-on-windows-11/>

2. Зарегистрировать (если не зарегистрирован) аккаунт на GitHub:

<https://github.com/>

рекомендация: аккаунт лучше именовать в формате: фамилия\_имя

3. Установить IDLE Python (версии 3.12) (не забыть поставить галочку "Add Python 3.12 to PATH").

<https://www.python.org/downloads/>

4. Установить <https://www.jetbrains.com/pycharm/> Community Edition

# Установка PyCharm

PyCharm

[What's New](#) [Features](#) [Learn](#)

[Pricing](#)

[Download](#)



Version: 2022.3.1  
Build: 223.8214.51  
28 December 2022

[System requirements](#)

[Installation instructions](#)

[Other versions](#)

[Third-party software](#)

## Download PyCharm

[Windows](#)

[macOS](#)

[Linux](#)

### Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

[Download](#)

[.exe](#) ▼

Free 30-day trial available

### Community

For pure Python development

[Download](#)

[.exe](#) ▼

Free, built on open-source



Get the Toolbox App to download PyCharm and its future updates with ease

# Установка IDLE



## Active Python Releases

For more information visit the Python Developer's Guide.

Activate Wind  
Go to Settings to a

# Особенности языка

- Интерпретируемый язык высокого уровня
- Динамическая типизация
- Автоматический сборщик мусора
- Поддержка различных парадигм программирования:
  - Процедурное программирование
  - Объектно-ориентированный подход,
  - Функциональное программирование

# Задания - Первые программы на Python

```
print("Hello world")
```

```
#-----
```

```
s = input("Как тебя зовут?")
```

```
print("Привет!")      # А что будет, если написать print("Привет", s)
```

```
print(s)
```

```
#-----
```

```
print(f"Привет, {s}!")  # А что будет, если написать {s=}
```

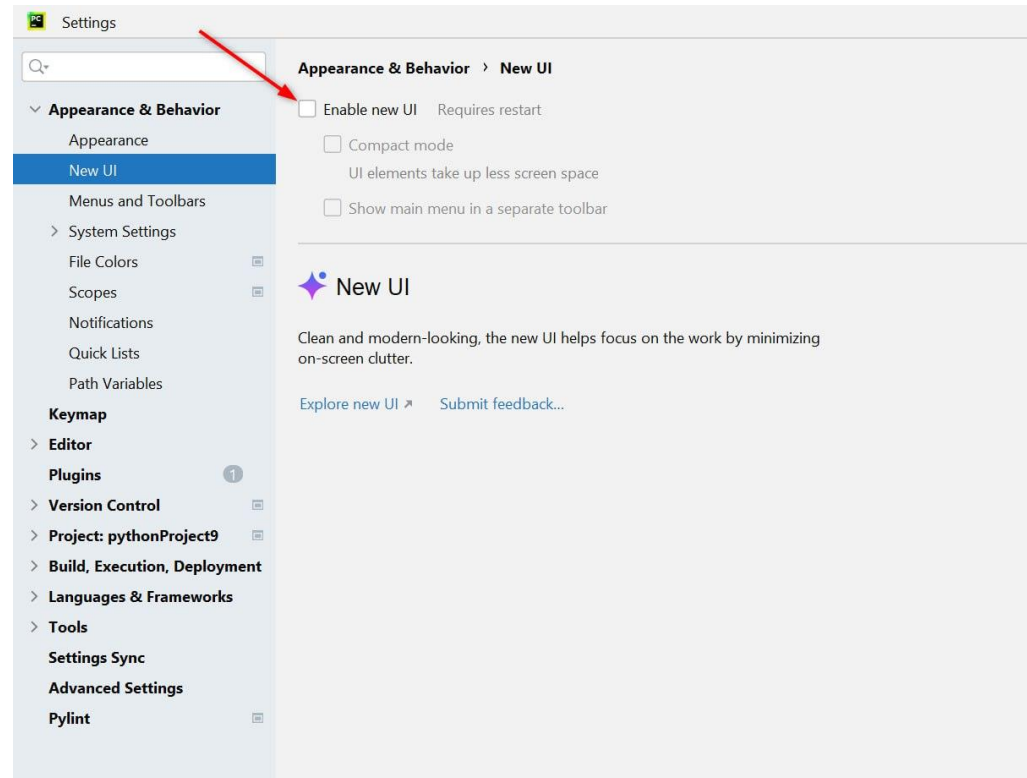
```
x = int(input("Сколько тебе лет?"))
```

```
print(f"Мне {x} лет")
```

```
# Выполните эти программы
```

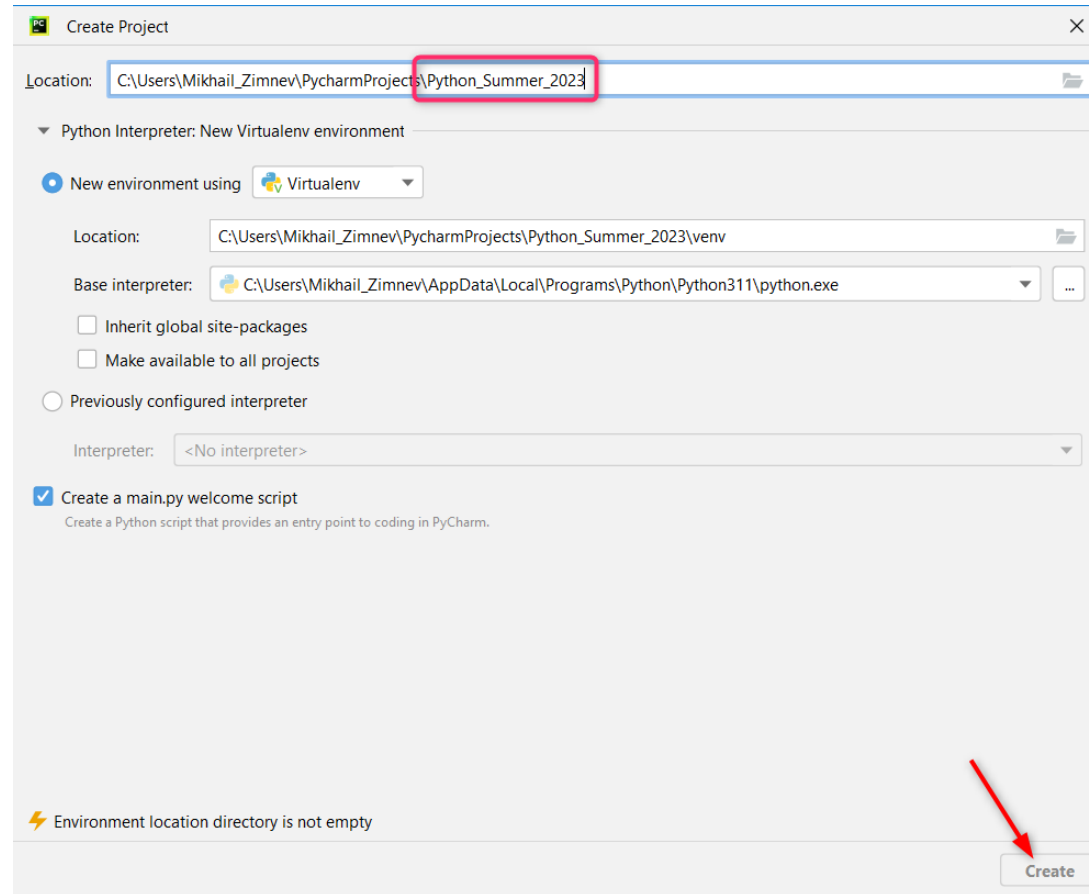
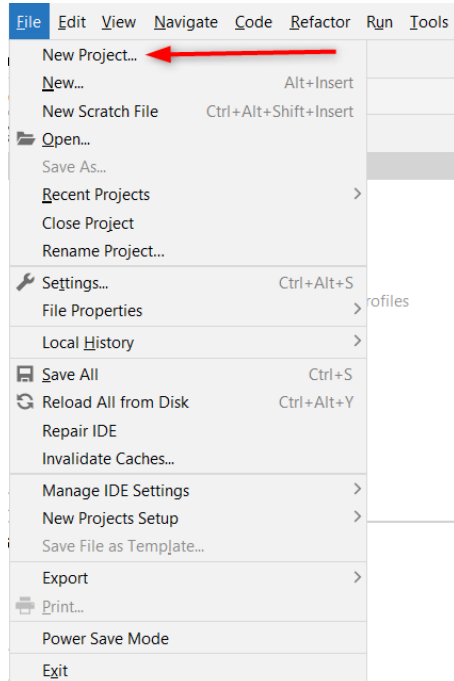
# Настройка UI

- Дальнейшие слайды относятся к традиционному интерфейсу. Поэтому убедитесь, что в настройках File/Settings/New UI настройка Enable new UI отключена.

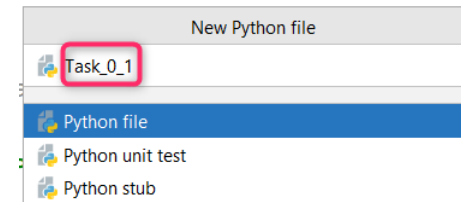
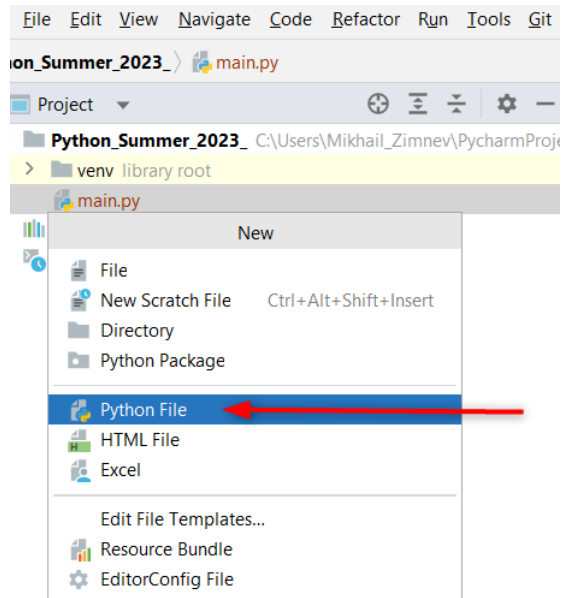
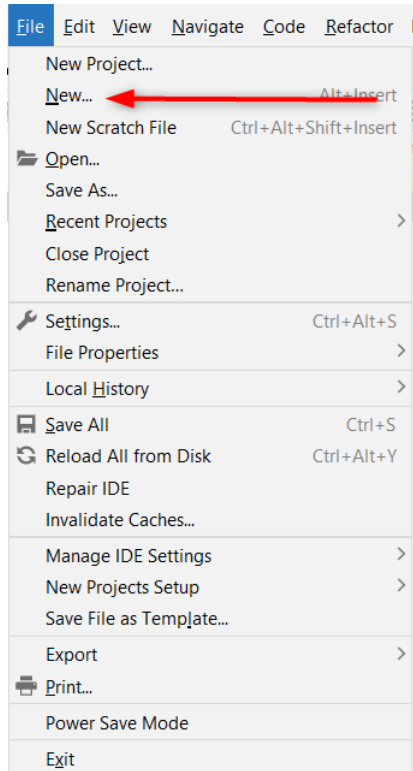




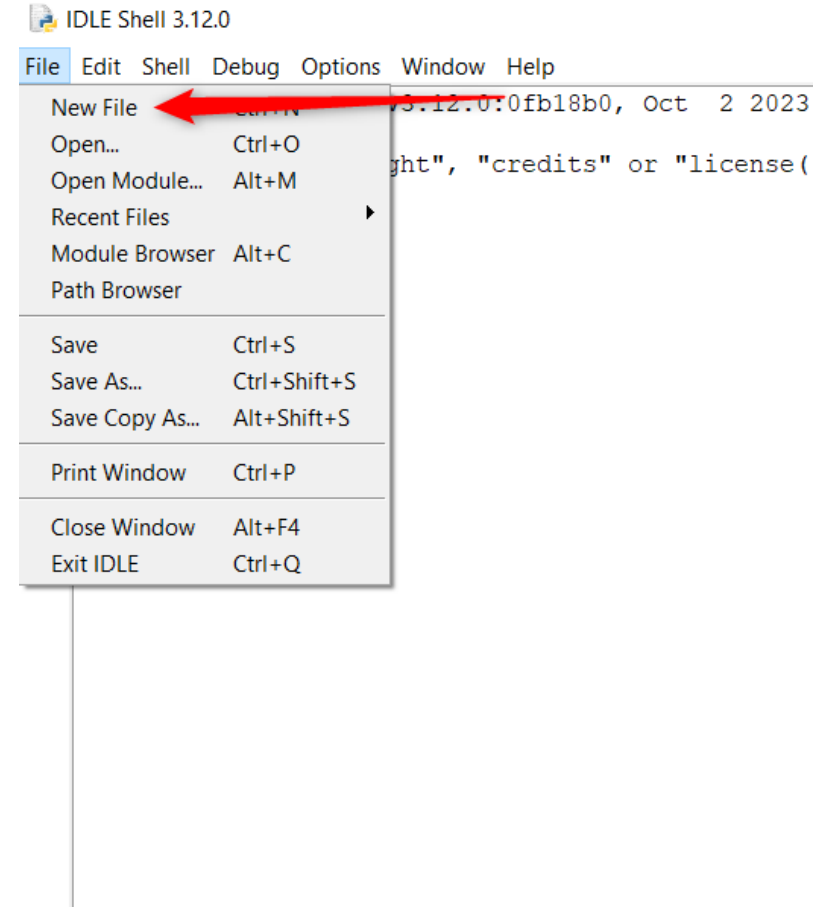
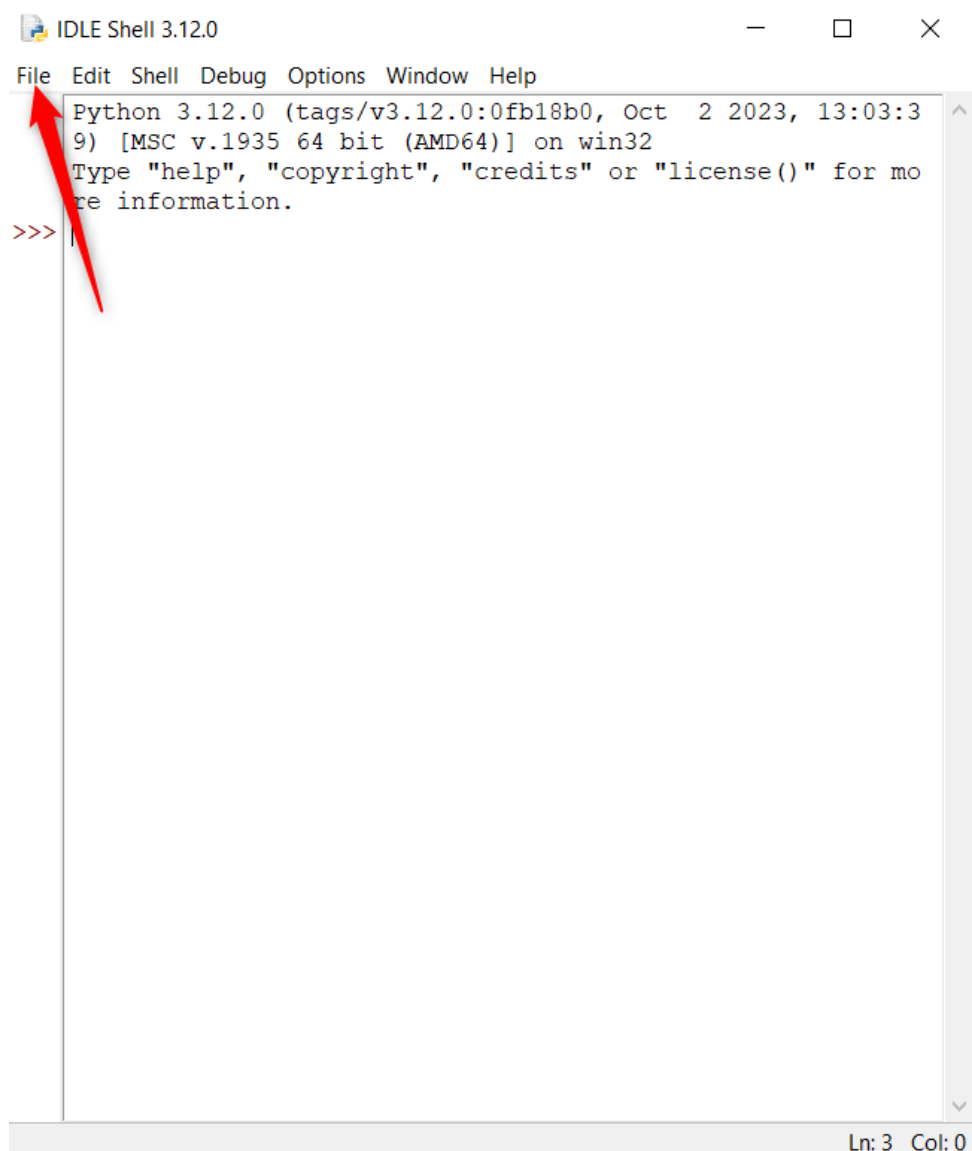
# Создание нового проекта в PyCharm



# Создание нового файла в PyCharm



# Создание нового файла в IDLE



# Первые программы на Python

```
a = 2
```

```
b = 3
```

```
print(a + b)
```

```
#-----
```

```
a = input()
```

```
b = input()
```

```
print(a + b)
```

```
#-----
```

```
a = float(input())    # а можно a = int(input())
```

```
b = float(input())
```

```
print(a, b, a + b)
```

# Как работает Python ?

1. Программа читается парсером и происходит анализ лексики.

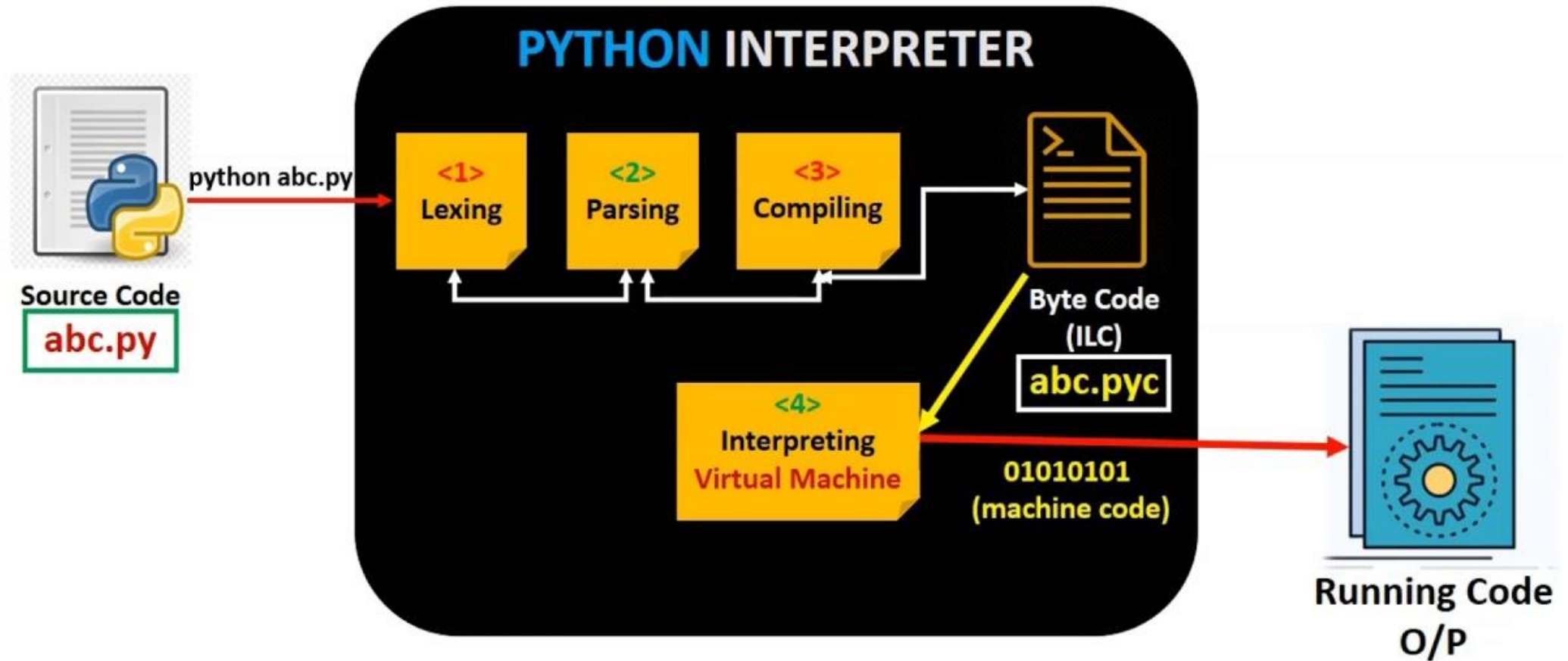
Parser - это анализатор синтаксиса.

В итоге получается набор лексем для дальнейшей обработки.

2. Затем парсером из инструкций происходит генерация структуры и формирования дерева синтаксического разбора- AST (Abstract Syntax Tree).

3. После этого компилятор преобразует AST в байт-код и отдает его на выполнение интерпретатору.

# Simulating Python Interpreter



# Для чего нам нужно знать про синтаксический разбор?

Если мы допускаем ошибки в грамматике кода то получаем **синтаксическую** ошибку.

- **SyntaxError: invalid syntax**

Наберите, плз, какую-нибудь чушь и попробуйте выполнить программу.

Интерпретатор. Плюсы и минусы?

# Задания

1. Напишите программу, которая запрашивает имя, потом запрашивает фамилию, а потом печатает: Привет, Петр Иванов
2. Напишите программу, которая запрашивает три имени и запоминает их в трех разных переменных, а потом печатает их в противоположном порядке в одной строке через двоеточие, например:

- Иван
- Петр
- Николай

А потом печатает Николай:Петр:Иван

Подсказка: при печати используйте `f" { } { } "`



# Философия Python

## import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one -- obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never. Although never is often better than \*right\* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

1. Красивое лучше, чем уродливое;
2. Явное лучше, чем неявное;
3. Простое лучше, чем сложное;
4. Сложное лучше, чем запутанное;
5. Плоское лучше, чем вложенное;
6. Разреженное лучше, чем плотное;
7. Читаемость имеет значение;
8. Особые случаи не настолько особые, чтобы нарушать правила;
9. При этом практичность важнее безупречности;
10. Ошибки никогда не должны замалчиваться;
11. Если не замалчиваются явно;
12. Встретив двусмысленность, отбрось искушение угадать;
13. Должен существовать один — и, желательно, только один — очевидный способ сделать это;
14. Хотя он поначалу может быть и не очевиден, если вы не голландец;
15. Сейчас лучше, чем никогда;
16. Хотя никогда зачастую лучше, чем прямо сейчас;
17. Если реализацию сложно объяснить — идея плоха;
18. Если реализацию легко объяснить — идея, возможно, хороша;
19. Пространства имён — отличная вещь! Давайте будем делать их больше!

# Зарезервированные слова

```
import keyword  
print(*keyword.kwlist)
```

Не используйте их в других целях!

# Как писать без ошибок ?

- Без ошибок писать пока не получится. Придется их устранять по ходу написания кода.
- Чтобы писать без ошибок нужно следовать правилам формальной грамматики языка.
- Кроме написания программ, мы еще будем учиться исправлять ошибки:
  - Синтаксические,
  - Алгоритмические,
  - Программные...
- Лучше учиться писать так, чтобы делать ошибок поменьше и чтобы их было легче находить
- Надо проверять работу программы **с различными данными**
- Всегда надо помнить, что программу кто-то будет читать, например, Вы через месяц. Надо постараться, чтобы программу этот кто-то понял с минимальными усилиями.

# Формальные языки

- Любой формальный язык, в том числе и Python, имеет три самые важные составляющие:
  - \* Операторы
  - \* Данные
  - \* Конструкции

Также в языках программирования часто присутствуют комментарии

# Рассмотрим как пример один из самых известных формальных языков

$(5 * 3 / (1+2))$

- В данном случае операторами являются
  - \* Оператор умножения
  - \* Оператор деления
  - \* Оператор сложения
  - \* Операторы группировки (скобочки)

Данными являются

- \* Число 5
- \* Число 3
- \* Число 1
- \* Число 2

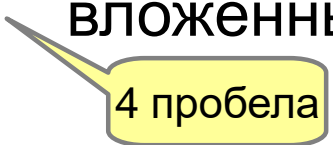
Сосчитайте с помощью Питона, чему равно это выражение

# ОСНОВЫ синтаксиса Python

- Программа - это заданная последовательность инструкций. Инструкции выполняются сверху вниз.
- Конец строки является концом инструкции (точка с запятой не требуется).
- Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым.
- Отступ должен быть одинаков в пределах вложенного блока. В Python принят отступ в 4 пробела.

# Отступы

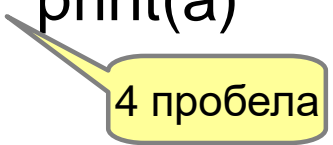
- Некоторые операторы языка (if, for, try и т.д.) требуют вложенные инструкции.
- Они в Python записываются в соответствии с одним и тем же шаблоном.
- Когда основная инструкция завершается двоеточием, за ней идет вложенный блок кода с отступом.
- Основная инструкция:  
    вложенный блок



4 пробела

```
if a > b:
```

```
    print(a)
```



4 пробела

# ОСНОВЫ синтаксиса Python

Допустимо записывать одну инструкцию в нескольких строках. Достаточно ее заключить в пару круглых, квадратных или фигурных скобок:

```
if (a == 1 and b == 2 and  
    c == 3 and d == 4): # Не забываем про двоеточие  
    print('spam' * 3)
```

Тело составной инструкции может располагаться в той же строке, что и тело основной, если тело составной инструкции не содержит составных инструкций.

```
if x > y: print(x)
```



# Оператор IF ELSE

```
x = int(input())
```

```
if x > 0:
```

```
    print("Положительное")
```

```
else:
```

```
    print("Отрицательное или 0")
```

```
# Чтобы ввести вещественное число, надо использовать float(input())
```

```
# Перепишите программу, чтобы она работала с вещественным числом
```

# Задание

Ввести число  $x$ .

Если оно  $< 0$ , то напечатать «Отрицательное».

Если оно  $> 0$ , то напечатать «Положительное»

Если оно  $== 0$ , то напечатать «Ноль»

# Задание

Ввести два числа:  $x$  и  $y$ .

Напечатать наибольшее из этих двух чисел.

# Задача 1-1

Ввести два числа  $x$  и  $y$ .

Напечатать сумму и произведение этих чисел  
(оператор  $+$  и  $*$ )

# Задача 1-2

Ввести два числа  $x$  и  $y$ .

Напечатать наибольшее из чисел  $x + y$ ,  $x - y$ ,  $x * y$ ,  $x / y$ ,  $x // y$

# Задача 1-3

Ввести два числа  $x$  и  $y$ .

Напечатать ВТОРОЕ ПО ВЕЛИЧИНЕ из чисел  $x + y$ ,  $x - y$ ,  $x * y$ ,  $x / y$ ,  $x // y$



Система контроля версий



# Полезная информация

- Н.Вирт Алгоритмы + структуры данных = программирование.  
[https://doc.lagout.org/science/0\\_Computer%20Science/2\\_Algorithms/Algorithms%20and%20Data%20Structures%20%28RU%29.pdf](https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Algorithms%20and%20Data%20Structures%20%28RU%29.pdf)
- О циклах разработки программного обеспечения  
[https://www.edsd.ru/ru/principy/cikl\\_razrabotki\\_po](https://www.edsd.ru/ru/principy/cikl_razrabotki_po)
- 11 самых популярных языков программирования в 2023 —  
зачем учить Python, JS и C++  
<https://eternalhost.net/blog/razrabotka/samye-populyarnye-yazyki-programmirovaniya>
- Популярные инструменты Python: библиотеки и фреймворки  
<https://eternalhost.net/blog/razrabotka/python-biblioteki>