

Python, установка Python, IDE, объекты и переменные, input() и print(), арифметические операции, условные конструкции

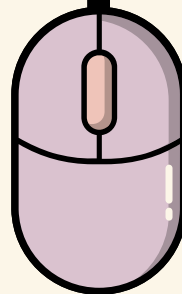
# Программирование в лингвистике.

## Лекция 1

Преподаватель - Верещагина Анна Дмитриевна

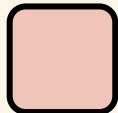
>>>>>

~~~~~  
.....





# Помним о них при программировании



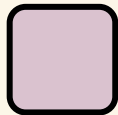
## CPU

Процессор. Получает команды на машинном языке и исполняет их.



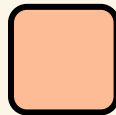
## GPU

Графический процессор. Основная функция – рендеринг графики. Полезен нейросетям.



## RAM

Оперативная память. Получает временные данные для обработки команд.



## ROM

Постоянная память. Хранит постоянные данные на диске.

# Как работает Python

1. Когда мы **программируем**, мы отдаем **CPU** или **GPU** **команды**, а **данные** кладем в **RAM**. Если нужно их **сохранить** – кладем в **ROM**.
2. Эти **команды** на ЯП превращаются в **машинный код**, единственно **понятный процессору**, благодаря «**переводчикам**».
3. Чем **ближе** ЯП к **машинному коду**, тем **быстрее** обрабатываются **команды**. Язык **C** ближе к **машинному коду**, чем Python, поэтому считается **быстрым**, но **сложным**.
4. **Python** написан на **C**. Таким образом, прежде чем **код на Python** дойдет до **процессора**, он будет переведен **несколько раз**. Поэтому он **не очень быстрый**, но **легкий** в освоении.



# Почему мы учим Python?

>>>>



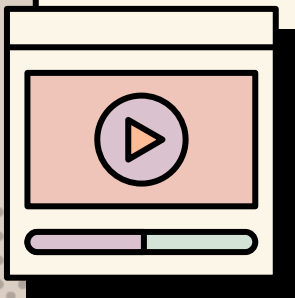
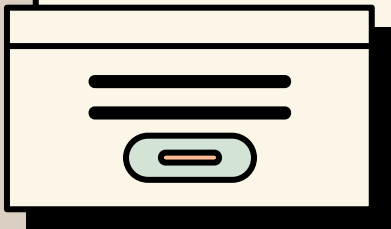
**Прост в освоении.** Согласно статье с [linkedin.com](#), относится к категории “relatively easy” вместе с JavaScript, CSS/HTML, Ruby и Scratch.



**Огромное количество библиотек, в том числе полезных для КЛ:** NLTK, spaCy, Gensim, Scikit-learn, PyTorch + HuggingFace и др.



**Лучший инструментарий для машинного обучения и нейросетей.** Занимает первое место по данному параметру согласно статье на [aiplusinfo.com](#).



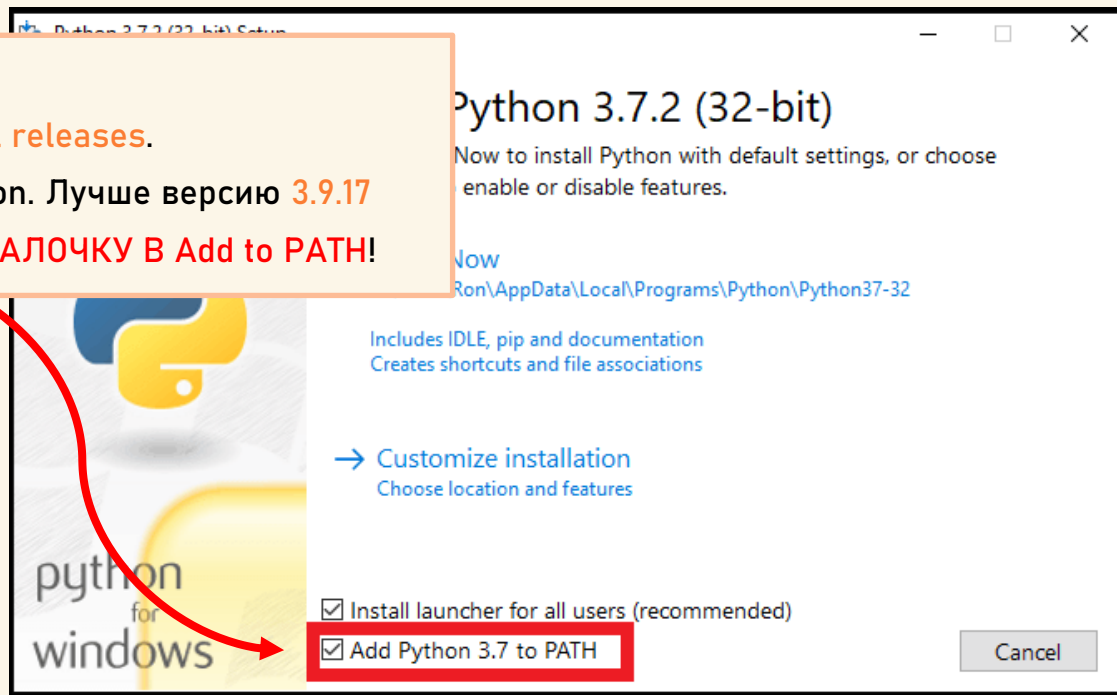
# Место Python в классификациях ЯП

| Классификация      | Что у Python             | Пояснение                                                                                                                                       |
|--------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Уровень ЯП         | Высокий                  | Чем <b>ниже</b> уровень, тем <b>ближе</b> ЯП к машинному коду.                                                                                  |
| Тип ЯП             | Интерпретируемый         | Python может выполнять код <b>построчно</b> , а компилируемые языки – только <b>целиком</b> .                                                   |
| Типизация данных   | Динамическая             | В Python не надо <b>предопределять тип</b> данных. Он <b>сам понимает</b> : строка это, число или др. В C надо обязательно обозначать этот тип. |
| Управление памятью | Автоматическое           | Не нужно решать, как <b>разместить</b> свои <b>данные</b> в <b>оперативной памяти</b> – Python делает это <b>сам</b> .                          |
| Парадигма ЯП       | Объектно-ориентированная | Изначально так, но Python может работать <b>в любой парадигме</b> . Весь мир для Python – <b>объекты разных типов</b> .                         |

# Установка Python

1. Заходим на сайт [python.org](https://python.org).
2. Переходим в [Downloads](#) -> [All releases](#).
3. Скачиваем дистрибутив Python. Лучше версию [3.9.17](#)
4. Устанавливаем. **ПОСТАВЬТЕ ГАЛОЧКУ В Add to PATH!**

Далее нам нужно будет установить **IDE** (интегрированную среду разработки) или пользоваться **IDLE** (на первых порах сойдет)

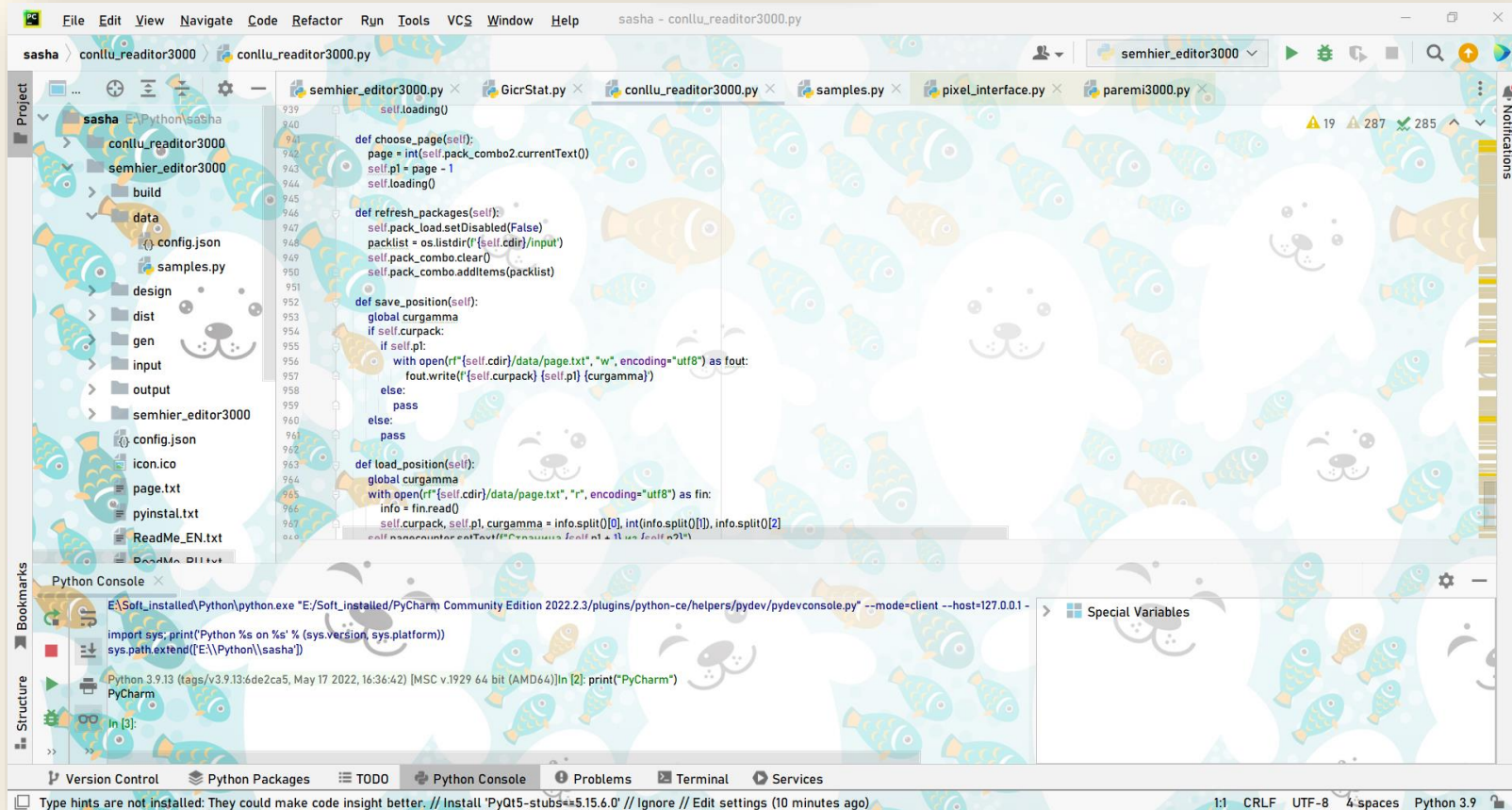




# ..... Сравнение основных IDE >>>>>~~~~~

|                    | Поддержка других ЯП | Установка плагинов | Виртуальные среды | Поддержка тетрадок | Простота | Размер на диске            | Скорость запуска |
|--------------------|---------------------|--------------------|-------------------|--------------------|----------|----------------------------|------------------|
| IDLE               | ✗                   | ✗                  | ✗                 | ✗                  | 1        | Размер дистрибутива Python | Быстро           |
| Google Colab       | ✓                   | ✗                  | ✓                 | ✓                  | 2        | Использует Google Disk     | Средне           |
| Visual Studio Code | ✓                   | ✓                  | ✓                 | ✓                  | 3        | 500 МБ                     | Быстро           |
| Jupyter Notebook   | ✗                   | ✗                  | ✗                 | ✓                  | 4        | 1 ГБ                       | Средне           |
| PyCharm            | ✗                   | ✓                  | ✓                 | ✗                  | 5        | 3.5 ГБ                     | Медленно         |

# PyCharm – кастомный интерфейс





# IDLE – устанавливается вместе с Python



```
semhier_editor3000.py - E:\Python\sasha\semhier_editor3000\semhier_editor3000.py (3.9.13)
File Edit Format Run Options Window Help

def change_path(self):
    prefpth = self.path
    path = QFileDialog.getOpenFileName(self, tips[self.guil][9],
                                      prefpth, "Text files (*.conllu *.txt)")[0] # Выберите файл с данными
    if path and os.path.exists(path):
        self.path = path
        self.pathline.setText(path)
    else:
        pass

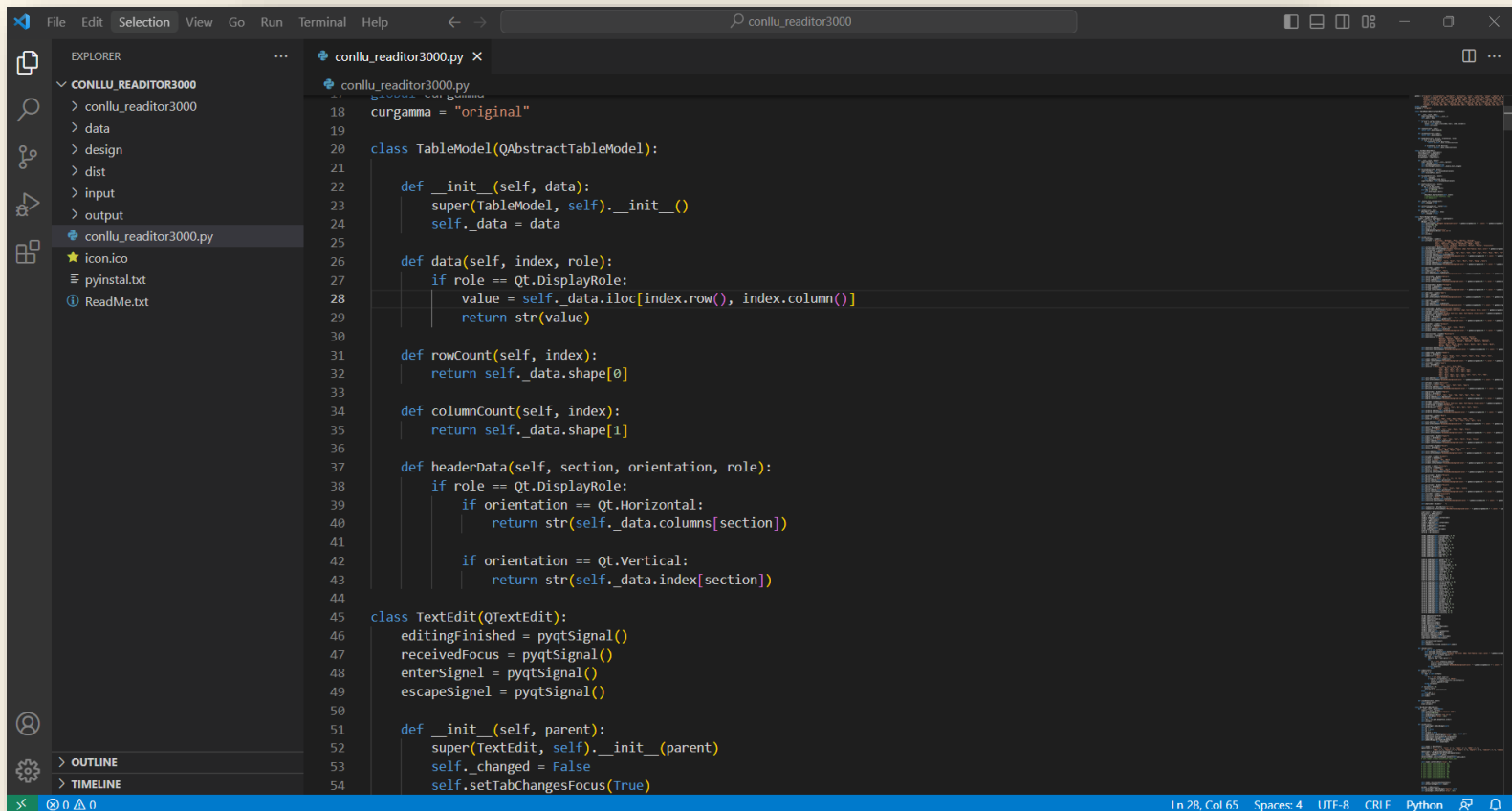
def generate(self):
    number = self.exspin.value()
    hier = os.listdir(rf"{self.cdir}/input")[0]
    color1, color2 = self.colcombol.currentText(), self.colcombo2.currentText()
    checked = self.classcheck.isChecked()
    s = Sampler(self.path, Hierarchy(f'{self.cdir}/input/{hier}', number, color1, color2, checked)) # путь к файлу
    s.harvest(rf"{self.cdir}/output/output_ex_{self.data_language}") # куда писать, без расширения - пишется жисон
    self.signal.emit()
    self.close()

class MainWinder(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("SemHier Editor 3000")
        self.resize(1500, 740)
        self.setWindowIcon(QIcon('icon.ico'))
        self.setFont(QFont('Arial', 14))
        self.w2 = None
        self.cdir = os.path.abspath(os.curdir)
        with open(rf"{self.cdir}/data/config.json", "r", encoding="utf8") as pager:
            dater = json.load(pager)
            self.guil = dater["GUI_language"]
            self.data_language = dater["data_language"]
        self.edited = False
        self.current_class = False
        self.settings = QSettings("Peck-Soft", "Semhier Editor 3000")

        self.resize(self.settings.value("size", QSize(1500, 740)))
        self.move(self.settings.value("pos", QPoint(50, 50)))
```

Ln: 1 Col: 0

# Visual Studio Code – дефолтный интерфейс





— □ ×











# Jupyter Notebook - интерфейс



 jupyter Gicr2AgePlot (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

        Code

```
In [ ]: dictionary = pd.read_excel("Словарь ГИКРЯ2 - 3.xlsx", engine="openpyxl", header=1)

In [ ]: n = 5 # Количество столбцов
```

Загружаем файл словаря для colab.

Наш словарь - это общая статистика корпусов, которую дает подсчет в самом ГИКРЯ. Нас интересуют в первую очередь годы рождения. Словарь выглядит так:

| возраст ВК  |                   | возраст ЖЖ  |                   |
|-------------|-------------------|-------------|-------------------|
| Рождение_ВК | Стат. Рождение_ВК | Рождение_ЖЖ | Стат. Рождение_ЖЖ |
| 1971        | 454194            | 1971        | 35519941          |
| 1972        | 502855            | 1972        | 39178663          |
| 1973        | 503431            | 1973        | 46050591          |
| 1974        | 564344            | 1974        | 54638600          |
| 1975        | 634772            | 1975        | 59840623          |
| 1976        | 668227            | 1976        | 68869619          |

Получение данных. В этом варианте кода сниппеты берутся прямо из colab. Корпусом оставляем ВК, так как ЖЖ еще не готов. Читаем файл сниппета в Excel формате, превращая все данные в DataFrame. Для красивого вывода также вынимаем из сниппета наименование паремии (первый случай вхождения).

```
In [ ]: # file_input = input("Введите имя файла: ")
        filepath = 'snippets.loreleiaethergmailcom_40954.xlsx'
```



>>>>

# Концепции Python



## CPython

Классический Python в С.  
Без тетрадок.

IDE: Visual Studio Code, Wing,  
PyCharm, IDLE и др.

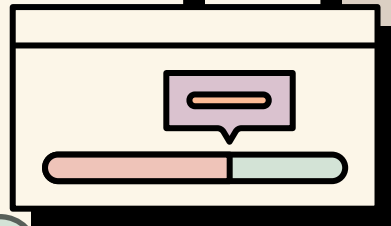


## IPython

Более интерактивная  
версия, использующая  
тетрадки.

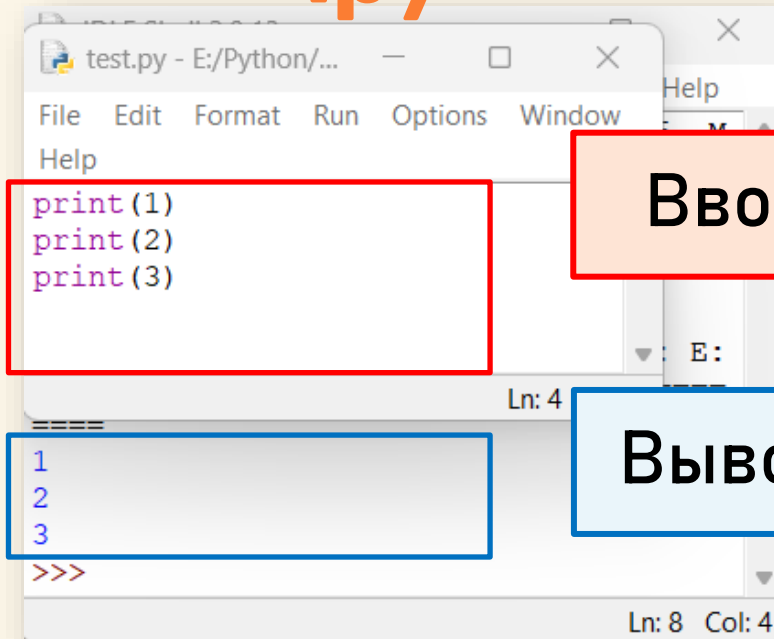
IDE: Jupyter Notebook,  
Google Colab, PyCharm  
Professional и др.

~~~~~  
.....



# Отличие .py-файлов от тетрадок

.py



The screenshot shows a text editor window titled 'test.py - E:/Python/...'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains three lines of Python code: `print(1)`, `print(2)`, and `print(3)`. These lines are enclosed in a red rectangular box. Below the code editor, the output is displayed as three separate lines: `1`, `2`, and `3`. These output lines are enclosed in a blue rectangular box. The status bar at the bottom indicates 'Ln: 8 Col: 4'.

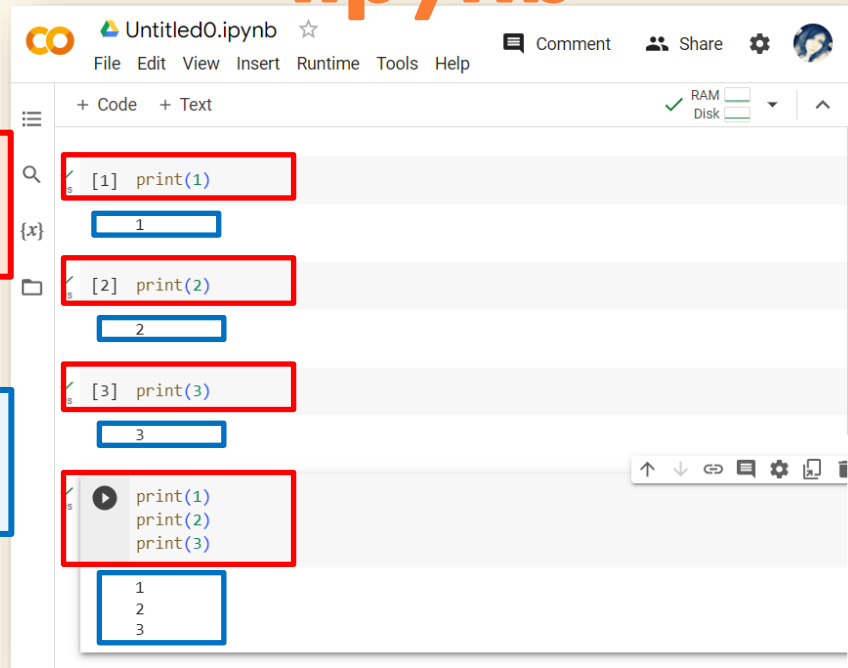
```
print(1)
print(2)
print(3)
```

```
1
2
3
```

Ввод

Вывод

.ipynb



The screenshot shows a Jupyter Notebook interface titled 'Untitled0.ipynb'. The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The interface displays three code cells, each enclosed in a red rectangular box. The first cell contains `print(1)` and has an output of `1` (enclosed in a blue box). The second cell contains `print(2)` and has an output of `2` (enclosed in a blue box). The third cell contains `print(3)` and has an output of `3` (enclosed in a blue box). At the bottom, there is a cell containing all three `print` statements, which has an output box containing `1`, `2`, and `3` stacked vertically (enclosed in a blue box). The status bar at the bottom right shows RAM and Disk usage.

```
[1] print(1)
```

```
1
```

```
[2] print(2)
```

```
2
```

```
[3] print(3)
```

```
3
```

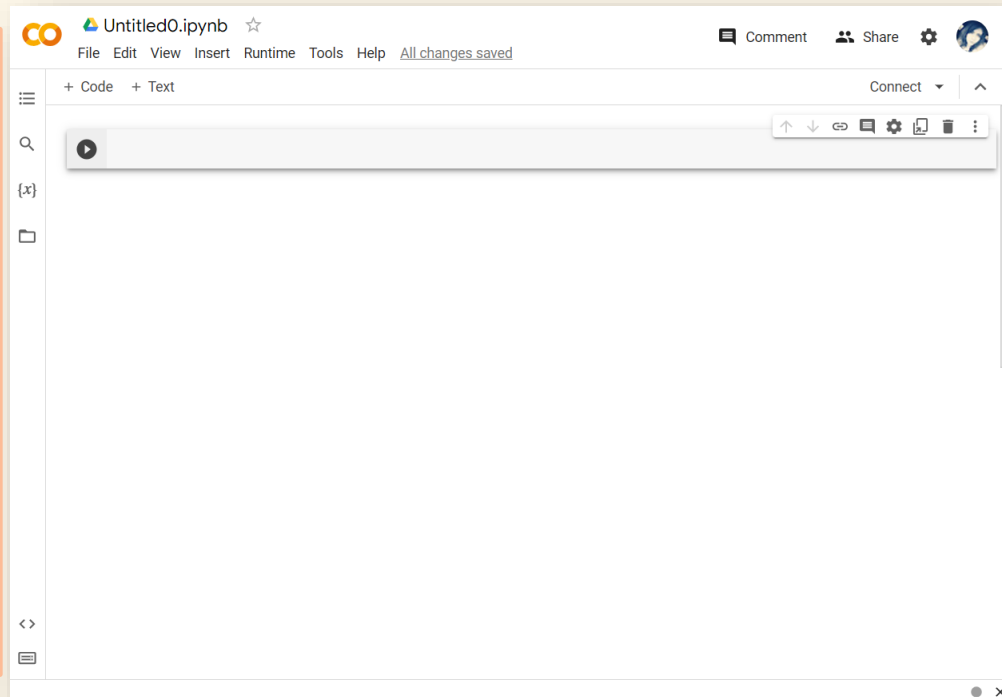
```
print(1)
print(2)
print(3)
```

```
1
2
3
```

# Работа в Google Colab

1. Авторизуемся в **Google**.
2. Заходим в **Google Диск**.
3. Создаем **папку** для работы с Python (можно не создавать, но так удобнее).
4. Правой кнопкой на пустом месте -> **Подключить другие приложения**.
5. Ищем **Google Colaboratory** и устанавливаем.
6. Создаем **объект тетрадки Colab**:

Правая клавиша мыши -> «Еще» -> «Google Colaboratory».



# Как выполнять задания в Colab

+ Code («b», когда никакая ячейка не активна): добавляет ячейку кода.

+ Text (можете установить свой shortcut): добавляет ячейку текста.

↑ («ctrl+M+K») – поднять ячейку выше.

↓ («ctrl+M+J») – опустить ячейку ниже.

▶ («ctrl+Enter») – исполнить код в выбранной ячейке.

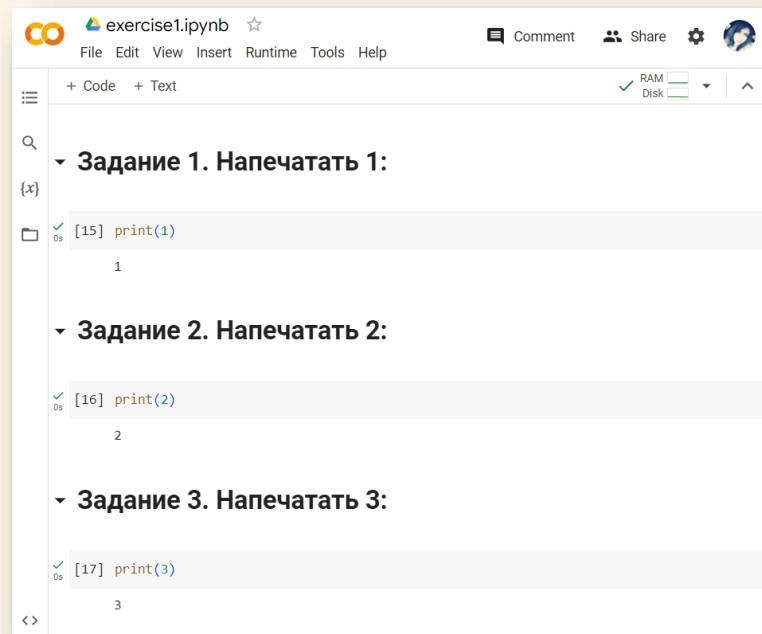
🗑 («ctrl+M+D») – удалить ячейку.

Runtime -> Run All («ctrl+F9») – исполнить код во всех ячейках последовательно.

File -> Rename – переименовать файл.

File -> Save – сохранить файл.

## Пример выполнения домашки



The screenshot shows a Google Colab notebook interface. At the top, the title bar says 'exercise1.ipynb' with a star icon. Below it are menu items: File, Edit, View, Insert, Runtime, Tools, Help. On the right, there are icons for Comment, Share, and a user profile. The notebook content area shows three code cells, each preceded by a dropdown arrow and a title:

- Задание 1. Напечатать 1:**  
[15] `print(1)`  
1
- Задание 2. Напечатать 2:**  
[16] `print(2)`  
2
- Задание 3. Напечатать 3:**  
[17] `print(3)`  
3

Each cell has a green checkmark and '0s' in the left margin, indicating successful execution. The output of each print statement is displayed below the code line.



# Установка Python-пакетов

## Понадобится для IDE Jupyter Notebook

Установка. Классический метод установки Python-пакетов через pip:

1. Открываем командную строку (**cmd** в меню **Пуск** или **win+R** и написать **cmd**).
2. Есть несколько вариантов установки пакетов. Если что-то не работает, возможно, стоит попробовать другой:

```
pip install [название пакета]
pip3 install [название пакета] для Linux
python pip install [название пакета]
py -m pip install [название пакета]
python -m pip install [название пакета]
```

Таким же образом устанавливаем Jupyter Notebook:

```
pip install jupyter
```

3. Дожидаемся установки пакета и надеемся, что он установился правильно.

# Запуск Jupyter Notebook

## ...А также используя .bat-файл

**Запуск в cmd.** Открываем **cmd** и последовательно вводим команды. Если не работает, можете найти рабочий вариант последней строки, как **справа**.

[индекс диска]:

```
cd [абсолютный путь к папке для работы]
jupyter notebook
```

```
C:\Users\user>E:
```

```
E:>>cd E:\Python\jupyter_notebook
```

```
E:\Python\jupyter_notebook>jupyter notebook
```

**Используем .bat-файл:**

1. Создайте в папке для работы **текстовый документ** с удобным названием (допустим, «Jupyter.txt»).
2. Откройте файл в любом текстовом редакторе и **впишите алгоритм** для командной строки.

Впишите из приведенных **один** рабочий:

```
jupyter notebook
python jupyter notebook
python -m jupyter notebook
py -m jupyter notebook
```

3. Измените расширение файла на **.bat**. Теперь Вы можете запустить Jupyter в два клика. **Будет открываться та папка, в которой Вы запустили файл.**

# Как выполнять задания в Jupyter

Insert -> Insert cell above («А») – Вставить ячейку кода **сверху**.

Insert -> Insert cell below («В») – Вставить ячейку кода **снизу**.

Cell -> Cell type -> ... – Выбрать **тип** ячейки.

«М» на выбранной неактивной ячейке кода – **превратить** в ячейку **с текстом**. «Y» – превратить ячейку текста в ячейку кода.

Edit -> Delete cells («D» **дважды** на выбранной неактивной ячейке) – **Удалить** ячейку.

Edit -> Move cell down/up – **Сдвинуть** ячейку вниз или вверх.

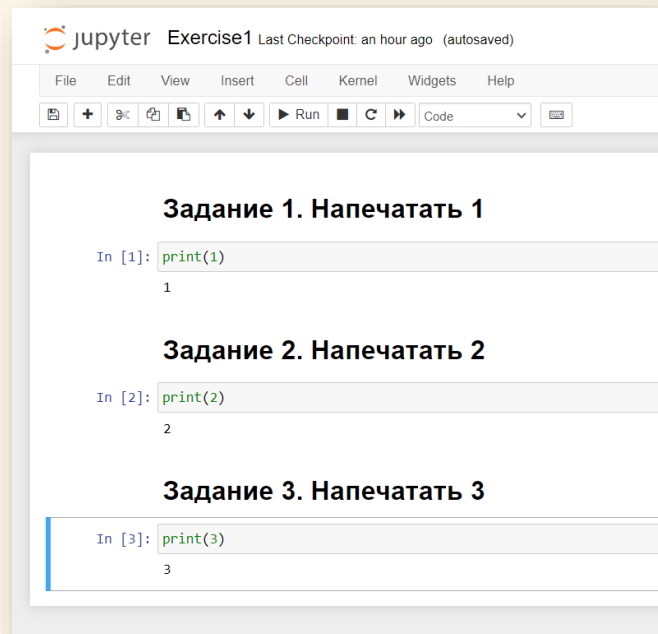
Cell -> Run cells («ctrl+Enter») – **Запустить** **выбранные** ячейки.

Kernel -> Restart & run all – **Запустить** **все** ячейки последовательно.

**Форматирование** в текстовой ячейке:

«# » перед текстом в разном кол-ве – меняет **размер заголовка**.

## Пример выполнения домашки





# Консоль VS Скрипт: режимы >>>>> ~~~~....

## Интерактивный

В консоли команды вводятся **по одной**.

Одна команда выполняется при нажатии на **Enter**, созданные таким образом **объекты запоминаются**.

Однако, код, написанный в консоли, на диск **не сохраняется**.

**Вывод:** консоль хороша для быстрых вычислений и презентации.

## Скриптовый

Код пишется **полностью** и **сохраняется** в постоянной памяти. При его запуске код выполняется **целиком**, вывод текста (print) и информация об ошибках появляются **в консоли**. Удобно, что во многих IDE такой код **можно запустить через debugger**.

**Вывод:** скриптовый режим – написание программ для многократного запуска.

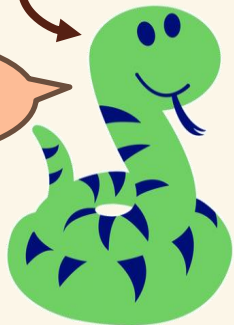
# Объекты Python

Ввод (кодим)

```
>>> number = 15  
>>> sett = {1, 2, 3}  
>>> string = "abc"
```

Вывод

Понял



Оперативная  
память

abc

3, 1, 2

15

Ввод

```
>>> number
```

Вывод

15



# Встроенные типы данных

Mutable

**ИЗМЕНЯЕМЫЕ**

- ✦ Списки (**list**):  
`>>> lst = [1, 2, 3]`
- ✦ Словари (**dict**):  
`>>> dct = {"ivanov": 5, "smirnov": 2}`
- ✦ Множества (**set**):  
`>>> sett = {"y", "h", "e"}`

Immutable

**НЕИЗМЕНЯЕМЫЕ**

- ✦ Числа (**int**, **float**, **complex**(мнимое число), **bool**):  
`>>> n = 1` и `fln = 1.23` и `compl = 2+3j` и `booll = True`
- ✦ Строки (**str**):  
`>>> string = "Hello world"`
- ✦ Последовательности байтов (**bytes**):  
`>>> bites = b'\xd0\x91'`
- ✦ Кортежи (**tuple**):  
`>>> tpl = («Hello world», 2)`

Мы не можем изменить отдельный элемент неизменяемого объекта. Так, мы не можем в целом числе (**int**) **1059** заменить цифру **5** на цифру **4**. Вместо этого нам придется перезаписывать данные целиком.

# Арифметические операции

Скобки работают как в математике!

Операция	Только с int	С float	Со строками
Сложение	$1 + 2 = 3$ (int)	$1.0 + 2 = 3.0$ (float)	"a" + "b" = "ab" (str)
Вычитание	$3 - 2 = 1$ (int)	$3.5 - 2.5 = 1.0$ (float)	Type Error!
Умножение	$2 * 2 = 4$ (int)	$2.5 * 2 = 5.0$ (float)	"a" * 3 = "aaa" (str)
Деление	$4 / 2 = 2.0$ (float)	$5 / 2.5 = 2.0$ (float)	Type Error!
Целочисленное деление	$9 // 2 = 4$ (int)	$10 // 7.5 = 1.0$ (float)	Type Error!
Остаток от деления	$10 \% 3 = 1$ (int)	$1 \% 0.5 = 0.0$ (float)	Type Error!
Возведение в степень	$2 ** 3 = 8$ (int)	$1.5 ** 2 = 2.25$ (float)	Type Error!

А если мы введем то же самое в консоль (с ответом), то результатом будет True.

# Операторы сравнения

Возвращают объект типа `bool`: `True` (истина) или `False` (ложь).

Операция	С числами	Со строками
Больше	<code>3 &gt; 1 -&gt; True</code>	<code>"b" &gt; "c" -&gt; False</code>
Меньше	<code>3 &lt; 1 -&gt; False</code>	<code>"a" &lt; "b" -&gt; True</code>
Больше или равно	<code>3 &gt;= 3 -&gt; True</code>	<code>"a" &gt;= "b" -&gt; False</code>
Меньше или равно	<code>1 &lt;= 3 -&gt; True</code>	<code>"a" &lt;= "c" -&gt; True</code>
Равно (не путать с "=")	<code>3 == 2 -&gt; False</code>	<code>"a" == "A" -&gt; False</code>
Не равно	<code>3 != 2 -&gt; True</code>	<code>"a" != "b" -&gt; True</code>
Используя «и»	<code>1 &lt; 3 and 6 &lt; 3 -&gt; False</code>	<code>"a" in "ab" and "b" in "ab" -&gt; True</code>
Используя «или»	<code>1 &lt; 3 or 6 &lt; 3 -&gt; True</code>	<code>"a" in "c" or "b" in "c" -&gt; False</code>
Используя «не»	<code>1 &lt; 3 and not 6 &lt; 3 -&gt; True</code>	<code>"a" in "ab" and not "b" in "ac" -&gt; True</code>

Скобки тоже работают.

`True` (1) и `False` (0) можно оперировать как числами:

`True + True = 2`  
`True - False = 1`  
`False * False = 0`



# Логические операторы

...И особенности их работы

Операнд 1	and / or	Операнд 2	Результат
1 < 3 (True)	and	2 < 3 (True)	True
2 < 3 (True)	and	3 < 1 (False)	False
3 < 1 (False)	and	2 < 3 (True)	False
1 < 3 (True)	or	2 < 3 (True)	True
2 < 3 (True)	or	3 < 1 (False)	True
2 < 1 (False)	or	3 < 1 (False)	False

# Условные конструкции

**Отступ!**

(4 пробела  
или tab)

elif, else  
необяза-  
тельны

Пример  
простой  
проверки:

## Общий шаблон

```
if [условие]:  
    [код]  
elif [другое условие]:  
    [код]  
elif [еще условие]:  
    [код]  
elif (сколько угодно раз):  
    ....  
else:  
    [код]
```

If [что-то]:  
[сделать что-то]

	Пояснение	Выполняется
if	если	если <b>True</b>
elif (else if)	иначе если	если предыдущее условие <b>False</b> , а это - <b>True</b>
else	иначе	Если все условия выше <b>False</b>

```
▶ a = 1  
b = 2  
if a == b:  
    print("1 = 2? Вряд ли")  
elif a > b:  
    print("1 > 2? Ну не")  
elif a < b:  
    print("1 < 2? Это верно")  
else:  
    print("Ну не знаю")
```

1 < 2? Это верно

# Еще примеры

Можно выполнять такие проверки, не прописывая условий (все равно получается bool):

Со  
строкой

```
a = ""
if a:
    print("a - это не пустая строка")
else:
    print("a - пустая строка")
```

a - пустая строка

```
[5] x = 5
    c = 0
    if x and c:
        print("x и c не равны 0")
    elif not c:
        print("c равен 0")
```

c равен 0

С целым  
числом



```
b = False
c = True
if c and b:
    print("b и c являются True")
elif c:
    print("c является True")
# Еще одна проверка
if c and not b:
    print("c является True, a b - False")
```

c является True  
c является True, a b - False

С bool

# Hello world, input() и print()

`print()` – функция «печать».

Выводит данные в консоль.

`input()` – функция «ввод».

В консоли запрашивает у пользователя ввести данные.

Всегда возвращает строку (str).

Пример с вводом:

```
input_data = input()
```

Hello world!

Простейший пример с выводом:

```
print("Hello world!")  
print(2023)
```

Hello world!  
2023

Совместим:

```
input_data = input("Поздоровайся: ")  
print("И тебе " + input_data)
```

Поздоровайся: привет!  
И тебе привет!

# Преобразование типов

Используя наименования типов объектов, можно преобразовывать их, чтобы в дальнейшем применять к ним необходимые функции или методы.

```
x = 3.0
print(type(x))
x = int(x)
print(x, type(x))
```

```
<class 'float'>
3 <class 'int'>
```

Число с плавающей точкой -> Целое число

```
[11] s = '6.3'
print(type(s))
s = float(s)
print(s, type(s))
```

```
<class 'str'>
6.3 <class 'float'>
```

Строка -> Число с плавающей точкой

Это нам сильно пригодится при использовании функции `input()`, всегда возвращающей строку.

```
x = input()
x += 2
```

5

TypeError



```
x = int(input())
print(x + 2)
```

5

7



# Пример красивого кода

```
user_name = input("Ваше имя: ") # Выбирайте названия объектов Python иллюстративно.
int = int(input()) '''Так делать нельзя. Избегайте совпадений имен ваших
объектов с типами, функциями, методами и др. Python или импортируемых библиотек.'''
user_age = int(input("Ваш возраст: ")) # При присвоении обе стороны отделяются пробелами.
result = (user_age + 5) // 2 ** 3 # Все арифметические операции отбиваются пробелами.
# Усл. конструкции, функции, классы, циклы рекомендуется отбивать пустой строкой.

if user_age > 20: # Рекомендуется использовать в качестве отступа не tab, а 4 пробела.
    print(user_name) # Скобки никогда не отбиваются пробелами.

'''Если конструкции, используемые в условном выражении составные, то их следует
занести в скобки:'''
if (user_age > 20) and (user_name == "Вася"):
    print(user_name, user_age) # После запятой ставится пробел.
```



>>>>

# Аббревиатуры

**IDE** = Integrated Development Environment. Интегрированная среда разработки, где мы программируем.

**CPU** = Central Processing Unit. Центральный процессор.

**GPU** = Graphics Processing Unit. Графический процессор.

**RAM** = Random Access Memory. Оперативная память.

**ROM** = Read Only Memory. Постоянная память.

**ЯП** = Язык Программирования.

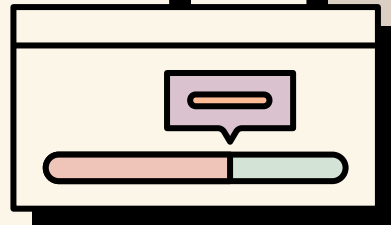
**Int** = Integer. Целое число.

**Float** = Floating point number. Число с плавающей точкой. Десятичная дробь.

**Str** = String. Строка.

**Bool** = Boolean. Булеан, логическая переменная.

~~~~~  
.....





# Спасибо за внимание!

Домашку и вопросы посылать сюда:  
Gmail: [admvereshchagina@gmail.com](mailto:admvereshchagina@gmail.com)  
Telegram: [https://t.me/lorelei\\_ether](https://t.me/lorelei_ether)