

< Teach  
Me  
Skills />

## Занятие 1.

# Введение в алгоритмизацию и экосистему python

# План занятия

- Знакомство
- Как будет проходить курс
- О программировании в целом
- О python
- Как запускать код
- Переменные
- Типы данных
- Работа с числами, операторы
- Работа со строками
- GIT
- Домашнее задание

# Давайте знакомиться

- Как вас зовут?
- Где учитесь / чем занимаетесь на данный момент?
- Есть ли какой-то опыт в IT?
- С чем хотите уйти после этого курса?

# Как будет проходить курс

1

Изучение python  
с нуля

2

Много практики

3

Дипломный  
проект и поиск  
работы



# Как будут проходить занятия

- 1** Проверка ДЗ и фидбэки
- 2** Повторение предыдущей темы
- 3** Теория
- 4** Практика
- 5** Объяснение ДЗ
- 6** Подведение итогов

# Правила работы в группе

- 1 Ты - Вы
- 2 Камеры
- 3 Общение
- 4 Вопросы

**1**

# О программировании в целом

# Что такое программирование?

**Программирование** – процесс создания компьютерных программ.

**Программирование** - это все что нас окружает в современном мире -  
все приложения, мессенджеры, кассы самообслуживания, которыми мы пользуемся  
каждый  
день

# Преимущества в ИТ

- Окружение из умных людей.
- Компании любят программистов. Их лелеют и пытаются удержать
- Возможность работать из любого места
- Возможность работать в любое время
- Возможность работать на себя
- Высокие зарплаты
- Нет такой строгой иерархии и отчетности



# Карьера в ИТ

- 1 Путь эксперта
- 2 Путь управленца
- 3 Путь стартапа

“

**Python** - высокоуровневый язык  
программирования, главной особенностью  
которого является читабельность и синтаксис, с  
помощью которого можно легко описать любую  
концепцию в нескольких строках кода

Гвидо Ван Россум



# Где используют python?

- 1 Веб-разработка
- 2 Machine Learning и AI
- 3 Big Data

# Преимущества Python

- Главная особенность Python в том что он один из самых выгодных языков для бизнеса
- Множество доступных сред разработки
- Низкий порог входа, простой синтаксис
- Быстрорастущий, есть библиотеки практически под все
- Множество вакансий

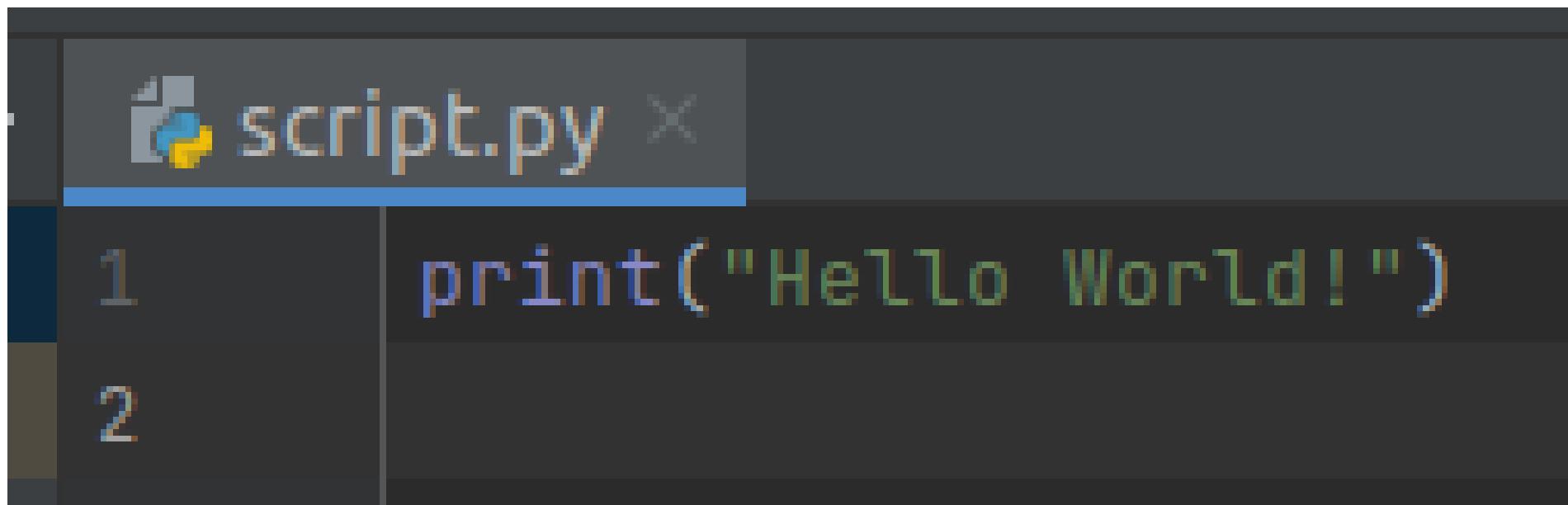
# Недостатки Python

- Низкая скорость работы кода (зависит от многих факторов)
- Не самый удобный язык для мобильных разработок
- Из-за гибкости типов данных потребление памяти Python не минимальное

2

Как запускать код

# Простейшая программа на Python



A screenshot of a code editor window titled "script.py". The window shows two lines of Python code:

```
1 print("Hello World!")  
2
```

# Как запустить код

- Писать код сразу в интерпретаторе
- Запустить скрипт в при помощи консоли
- Запустить код при помощи pycharm
- Выполнить код в jupyter notebook

# Команды ввода-вывода

**print()** - команда для вывода значений на экран

**input()** - команда для ввода данных из командной строки. Ввод данных завершается по нажатию клавиши Enter

3

# Переменные и типы данных

# Переменные

**Переменная** – поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной

**Пример объявления:**

a = 5

b = "Text"

# Что такое тип данных

**Тип данных** – множество значений и операций над этими значениями

`type()` - функция, при помощи которой можно узнать тип данных

**Пример использования:**

`a = 5`

`type(a)`

Вывод: `<class 'int'>`

# Основные типы данных

**числа:**

**int** - целые числа (1, 650, 10000000)

**float** - числа с плавающей точкой (0.1, 0.25, 378.98091)

**str** - строки ("text", "1234567890")

**bool** - логические переменные (True или False)

# Правила именования переменных

- Имя должно начинаться с **буквы или \_**
- Имя **не может начинаться с цифры**
- Имя может содержать только буквы английского алфавита, цифры и нижние подчеркивания (**A-z, 0-9, \_**)

## Примеры

name = "Lena"

age = 24

first\_name = "Lena"

# Соглашение для именования переменных

- При наличии нескольких слов используем `_` в качестве разделителя (`first_name`)
- Не использовать для переменных буквы верхнего регистра (`FirstName`)
- Использовать осознанные имена для названия переменных (`a = "Lena"` -> `first_name = "Lena"`)
- Не использовать ключевые слова для названия переменных (`print = 1`)

# Имена переменных и ключевые слова

## Хорошо

summ  
summ\_of\_variables  
\_new\_sum  
is\_payable  
\_\_another\_variable

## Плохо

Summ  
summOfVariables  
New\_sum  
isPayable  
\_\_anotherVariable

## Нельзя

Sum%t  
&ummOfVariable  
42New\_sum  
isP\*yable  
\_\_anotherV@riable

4

# Работа с числами

# Работа с числами (часть 1)

## Арифметические операторы в Python

Оператор	Описание	Примеры
+	Сложение - суммирует значения слева и справа от оператора	15 + 5 в результате будет 20 20 + -3 в результате будет 17 13.4 + 7 в результате будет 20.4
-	Вычитание - вычитает правый operand из левого	15 - 5 в результате будет 10 20 - -3 в результате будет 23 13.4 - 7 в результате будет 6.4
*	Умножение - перемножает operandы	5 * 5 в результате будет 25 7 * 3.2 в результате будет 22.4 -3 * 12 в результате будет -36

# Работа с числами (часть 2)

Оператор	Описание	Примеры
/	Деление - Делит левый операнд на правый	15 / 5 в результате будет 3 5 / 2 в результате будет 2 (В Python 2.x версии при делении двух целых чисел результат будет целое число) 5.0 / 2 в результате будет 2.5 (Чтобы получить "правильный" результат хотя бы один операнд должен быть float)
%	Деление по модулю - Делит левый операнд на правый и возвращает остаток.	6 % 2 в результате будет 0 7 % 2 в результате будет 1 13.2 % 5 в результате 3.2
**	Возведение в степень - возводит левый операнд в степень правого	5 * 5 в результате будет 25 7 * 3.2 в результате будет 22.4 -3 * 12 в результате будет -36
//	Целочисленное деление - Деление в котором возвращается только целая часть результата. Часть после запятой отбрасывается.	12 // 5 в результате будет 2 4 // 3 в результате будет 1 25 // 6 в результате будет 4

# Работа с числами (часть 3)

## Операторы сравнения в Python

Оператор	Описание	Примеры
<code>==</code>	Проверяет равны ли оба операнда. Если да, то условие становится истинным.	<code>5 == 5</code> в результате будет <code>True</code> <code>True == False</code> в результате будет <code>False</code> <code>"hello" == "hello"</code> в результате будет <code>True</code>
<code>!=</code>	Проверяет равны ли оба операнда. Если нет, то условие становится истинным.	<code>12 != 5</code> в результате будет <code>True</code> <code>False != False</code> в результате будет <code>False</code> <code>"hi" != "Hi"</code> в результате будет <code>True</code>
<code>&lt;&gt;</code>	Проверяет равны ли оба операнда. Если нет, то условие становится истинным.	<code>12 &lt;&gt; 5</code> в результате будет <code>True</code> . Похоже на оператор <code>!=</code>
<code>&gt;</code>	Проверяет больше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	<code>5 &gt; 2</code> в результате будет <code>True</code> . <code>True &gt; False</code> в результате будет <code>True</code> . <code>"A" &gt; "B"</code> в результате
<code>&lt;</code>	Проверяет меньше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	<code>3 &lt; 5</code> в результате будет <code>True</code> . <code>True &lt; False</code> в результате будет <code>False</code> . <code>"A" &lt; "B"</code> в результате будет <code>True</code> .

# Работа с числами (часть 4)

Оператор	Описание	Примеры
<b>&gt;=</b>	Проверяет больше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	$1 >= 1$ в результате будет True. $23 >= 3.2$ в результате будет True. $"C" >= "D"$ в результате будет False.
<b>&lt;=</b>	Проверяет меньше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	$4 <= 5$ в результате будет True. $0 <= 0.0$ в результате будет True. $-0.001 <= -36$ в результате будет False.

# Операторы, операнды и выражения

- **Операторы** - символы, которые представляют вычисления ( $a + b$ )
- **Операнды** - значения, к которым применяется оператор ( $a + b$ )
- **Выражение** - комбинация операторов и operandов ( $x + 17$ )

# Арифметические операции со сравнением

Вместо  $n = n + 1$  можно написать  $n += 1$ , а также:

$n -= 1$  равно  $n = n - 1$

$n *= 2$  равно  $n = n * 2$

$n /= 2$  равно  $n = n / 2$

$n // 2$  равно  $n = n // 2$

$n %= 2$  равно  $n = n \% 2$

$n **= 2$  равно  $n = n ** 2$

# Порядок операций

Если в выражении встречается больше, чем один оператор, то порядок вычисления зависит от правил старшинства (rules of precedence). Для математических операций, Python следует математическим соглашениям. **PEMDAS** (расположены от наивысшего приоритета с низкому)

- **Parentheses** - Скобки
- **Exponentiation** - Возведение в степень
- **Multiplication** - Умножение
- **Division** - Деление
- **Addition** - Сложение
- **Subtraction** - Вычитания

Операторы с одинаковым приоритетом вычисляются слева-направо

# Работа с числами - итоги

- При делении целых чисел получается float
- Для того чтобы преобразовать float в int используется выражение `int(a)` - это называется **приведение к типу**
- Выполнение операции сравнения возвращает **boolean (True/False)**
- Для математических операций над числами python следует математическим соглашениям **о приоритете операций**

5

# Строки

# Строки

**Строка** - последовательность символов, заключенная в кавычки. Неизменяемый упорядоченный тип данных

## Способы объявления:

```
a = "abc"
```

```
a = 'abc'
```

```
a = """
```

Многострочный текст  
заключается в тройные кавычки

```
"""
```

```
print(a)
```

Многострочный текст  
заключается в тройные кавычки

# Экранирование

Строка может содержать кавычки или апостроф:

```
a = "том's"  
a
```

"том's"

```
a = 'Текст в "кавычках"'  
a
```

'Текст в "кавычках"'

```
a = "А здесь используется \"экранриование"  
a
```

'А здесь используется "экранриование'  
заключается в тройные кавычки

# Экранирование

Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры, например \n - перевод строки

```
a = "А здесь используется \nперенос на новую строку"  
print(a)
```

А здесь используется  
перенос на новую строку

Подробнее: <https://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html>

# Форматирование строк

## Метод format:

```
name = "Lena"  
surname = "Deykun"  
string = "Hi, {name} {surname}".format(name=name, surname=surname)  
print(string)
```

Hi, Lena Deykun

## f-string (рекомендуемый):

```
name = "Lena"  
surname = "Deykun"  
string = f"Hi, {name} {surname}"  
print(string)
```

Hi, Lena Deykun

# Как работать со строками

- Конкатенация (сложение)
- Дублирование (\*)
- Получить длину строки (`len(str_1)`)
- Получить символ по индексу `string[0]`, `string[-2]`
- Извлечение среза `string[0:6]`, `string[0:6:2]`

## Другие методы

- `s.find()`
- `s.index()`
- `s.replace(" ", "")`
- `s.isalpha()`
- `s.capitalize()`
- `s.lower()`

# Кодировка

**Кодировка** - правила перевода одного набора символов в другой

## Самые популярные кодировки:

**ASCII** - таблица кодировки символов, в которой каждой букве, числу и знаку соответствует определенное число. В стандартной таблице ASCII 128 символов, пронумерованных от 0 до 127. В них входят латинские буквы, цифры, знаки препинания и управляемые символы.

**Unicode** - стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира. В настоящее время стандарт является преобладающим в Интернете.

**6**

**GIT**

# Что такое **git**

- **git** - система управления версиями
- **github** - облачная платформа для хостинга ИТ проектов и совместной разработки

## Зачем?

- сохранение новых версий файлов
- возможность откатиться на старую версию файла
- совместная разработка
- хранение кода проекта в облаке (на случай потери локальных данных)

# Как работать с git

- зарегистрироваться на GitHub
- установить git на вашу ОС
- зайти в нужную папку, открыть консоль

## Настройка git:

- `git config --global user.email "you@example.com"`
- `git config --global user.name "Your Name"`

# Как работать с git

- **git init** - инициализация репозитория
- **git checkout -b main** - переключиться на ветку main
- **git status** - показывает проиндексированные файлы
- **git add имя\_файла** - проиндексировать файл (сделать так чтобы изменения попали в текущий коммит)
- **git commit -m "Init commit"** - закоммитить изменения
- **git remote add origin https://github.com/username/techmeskills.git** - подключиться к удаленному репозиторию на GitHub
- **git push -u origin main** - загрузить содержимое локального репозитория на GitHub

# Домашнее задание

7

# Домашнее задание (часть 1)

**Часть 1:** научиться запускать python код (установить себе python и любой удобный редактор кода, например pycharm)

**Часть 2:** создать github репозиторий, загрузить туда ДЗ, поделиться со мной (elenadeyukun), а также написать в чат свой никнейм и ФИО

**Часть 3:**

- есть числа  $a$  и  $b$ , которые вводит пользователь. Программа последовательно выводит на экран их сумму, разность, произведение,  $a$  в степени  $b$  и целочисленное деление  $a$  на  $b$

# Домашнее задание (часть 2)

- расставить скобки так чтобы значение выражений не поменялось

`x = 17 / 2 * 3 + 2`

`x = 2 + 17 / 2 * 3`

`x = 19 % 4 + 15 / 2 * 3`

`x = (15 + 6) - 10 * 4`

`x = 17 / 2 % 2 * 3**3`

- дана строка `string="abcdefghi"`, нужно преобразовать ее таким образом чтобы получить

- "Abcdefghi"
- "ABCDEFGHI"
- "abc"
- "bdfh"

**Спасибо за внимание!**