

# PYTHON | БЕЗ ВОДЫ

Практический курс по Python


# **#Блок 2 | Ветвления**

Для быстрого старта

# О курсе

- ▶ Это ПРАКТИЧЕСКИЙ КУРС по программированию
- максимальная польза от курса в самостоятельном решении наибольшего числа задач
- вопросы на занятиях – самое ценное для учеников
- разбор задач учеников НА ЗАНЯТИЯХ
- разбор вопросов по задачам НА ЗАНЯТИЯХ

# Регламент

- ▶ Длительность одного занятия 90 минут (может +)
  - ▶ 50% теория / 50% практика
  - ▶ Регулярность занятий 1 раз в неделю
  - ▶ Проходим блоками
  - ▶ Занятия по скайпу
  - ▶ Запись занятий
  - ▶ Видео доступно на youtube в закрытом плейлисте
- 

# #1 | Про bool тип

Для быстрого старта

# Азбука

## ▶ Что такое bool тип:

- примитивный тип данных, принимающий два возможных значения, иногда называемых *истиной (True)* и *ложью (False)*

## ▶ Применяется:

- задачи на ветвления
- задачи на циклы
- задачи на функции
- задачи на строки
- задачи на ООП
- задачи на файлы
- прочие задачи

# Азбука

## ▶ Что такое истина (True):

- утверждение, которое считается правдой
- ✓ *Мы учим Python*
- ✓  $7 == 7$

## ▶ Что такое ложь (False):

- утверждение, которое считается ложью
- ✓  $7 == 8$
- ✓ *На дворе 2033 год*



# Джентельменский набор для быстрого старта

## ▶ Типы данных:

- Целые числа (int)
- Вещественные числа (float)
- **Логические (bool)**
- Строки (string)
- Списки (list)
- Кортежи (tuple)
- Словари (dict)
- Файлы (file)



# Джентельменский набор для быстрого старта

## ► Изменение типов данных:

- `int(x)`
- `float(x)`
- **`bool(x)`**
- `string(x)`
- `list(x)`
- `tuple(x)`
- `dict(x)`



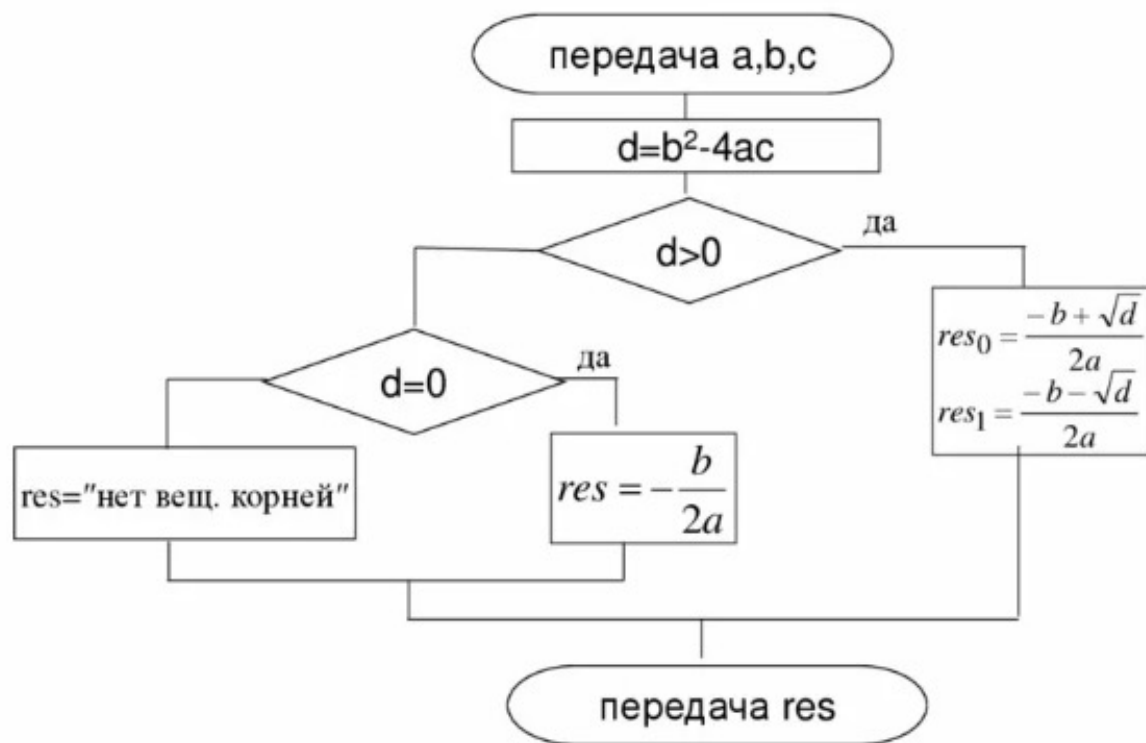
# Азбука

## ▶ Что делает `bool(x)`:

- 0 -> False
- 0.0 -> False
- "" -> False
- [] -> False
- () -> False
- 1, 2, 88, -100 -> True
- 6.9, -90.66 -> True
- '0' -> True
- [0] -> True
- (0) -> True

# Булев тип в блок-схемах

## Блок-схема алгоритма нахождения вещественных корней квадратного уравнения



► Как теперь выполняется программа?

# Как сделать ветвление

- ▶ Оператор if:

```
if <условие>:  
    <действия 1>
```



# Как сделать ветвление

- ▶ Оператор if-else:

```
if <условие>:  
    <действия 1>  
else:  
    <действия n>
```

- else – необязательный оператор

# Как сделать ветвление

## ▶ Оператор if-elif-else:

```
if <условие>:  
    <действия 1>  
elif <условие>:  
    <действия 2>  
elif <условие>:  
    <действия 3>  
...  
else:  
    <действия n>
```

- else – необязательный оператор

# Как сделать ветвление

► Пример:

```
cost = 1500
if cost < 1000:
    print("Скидок нет.")
elif cost < 2000:
    print("Скидка 2%.")
elif cost < 5000:
    print "Скидка 5%.")
else:
    print("Скидка 10%.")
```

# Как сделать ветвление

## ▶ Вложенные выражения if-elif-else:

```
# Запрашиваем у пользователя число
num = float(input("Введите число: "))
# Сохраняем нужный нам результат
if num > 0:
    # Определяем, какое прилагательное нам стоит использовать
    adjective = " "
    if num >= 1000000:
        adjective = " очень большое "
    elif num >= 1000:
        adjective = " большое "

    # Сохраняем в переменную положительные числа с правильным прилагательным
    result = "Это" + adjective + "положительное число"
elif num < 0:
    result = "Это отрицательное"
else:
    result = "Это ноль"

# Выводим результат
print(result)
```

# Внутри ветвления

## ▶ Внутри if:

- одно или несколько условий

*if (a>2) and (b < 100) or (c%19 == 0):*

- отступ (tab) или четыре пробела

*if (a>2):*

*print('a > 2')*

- преобразование условия в логическое выражение (True или False)

# Азбука

## ▶ Что делает if (x):

- 0 -> False
- 0.0 -> False
- "" -> False
- [] -> False
- () -> False
- 1, 2, 88, -100 -> True
- 6.9, -90.66 -> True
- '0' -> True
- [0] -> True
- (0) -> True

# Внутри ветвления

## ► Внутри if:

Оператор отношения	Значение
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
==	Равно
!=	Не равно

- двойные неравенства  $10 \leq x < 1000$
- двойные неравенства  $10 < x \geq 1000$

# Азбука

## ▶ Алгебра логики:

раздел математической логики, в котором изучаются логические операции над высказываниями

## ▶ Логические операции:

- |                    |                    |
|--------------------|--------------------|
| 1. Отрицание       | <code>not()</code> |
| 2. Конъюнкция      | <code>AND</code>   |
| 3. Дизъюнкция      | <code>OR</code>    |
| 4. Импликация      | <code>&lt;=</code> |
| 5. Эквиваленция    | <code>==</code>    |
| 6. Исключающее ИЛИ | <code>XOR</code>   |

## ▶ Таблица истинности:

это таблица, описывающая логическую функцию, а именно отражающую все значения функции при всех возможных значениях её аргументов

- на каждую операцию своя таблица
- должна быть в голове



# Азбука

## ► Инверсия / отрицание:

- $\text{not } (x)$
- $\text{not } x$

X	not X
0	1
1	0

# Азбука

► Конъюнкция:

- AND / Логическое И

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



# Азбука

## ► Дизъюнкция:

- OR / ИЛИ

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1



# Азбука

## ► Эквиваленция:

- $==$  / эквивалентность / равенство

X	Y	$X == Y$
0	0	1
0	1	0
1	0	0
1	1	1



# Азбука

► Импликация:

- $\leq$  /  $\rightarrow$  / следование

X	Y	$X \rightarrow Y$
0	0	1
0	1	1
1	0	0
1	1	1



# Азбука

## ► Исключающее ИЛИ (XOR):

- $\wedge$

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

# Внутри ветвления

## ► Приоритетность логических операций:

1. Выражение в скобках `()`
2. Отрицание `not()`
3. Конъюнкция `AND`
4. Дизъюнкция `OR`
5. Импликация `<=`
6. Эквиваленция `==`

# Внутри ветвления

## ► Популярные логические операторы:

- |                 |                                 |
|-----------------|---------------------------------|
| 1. Отрицание    | <code>not()</code>              |
| 2. Конъюнкция   | <code>AND</code>                |
| 3. Дизъюнкция   | <code>OR</code>                 |
| 4. Эквиваленция | <code>==</code>                 |
| 5. Комбинации   | <code>x OR y AND not (z)</code> |

# Сложности ветвлений

► Сложность задач на ветвления == сложность правильно написать условие

- условие нужно оптимизировать – укорачивать
- возможно стоит писать функции проверки (или использовать готовые в т.ч. из библиотек)
- искать наиболее оптимальный вариант проверки (с использованием других типов данных)
- думать нужны ли вложенные условия
- думать нужны ли `elif` и `else`
- правильно использовать `else`

# Джентельменский набор для быстрого старта

- ▶ Тернарный оператор

[если истина] if [выражение] else [если ложь]

- ▶ пример:

```
x = False if age < 18 else True
```



# Джентельменский набор для быстрого старта

## ► Оператор **match** (python >= 3.10)

match выражение:

```
case шаблон_1:  
    действие_1  
case шаблон_2:  
    действие_2  
.....  
case шаблон_N:  
    действие_N  
case _:  
    действие_по_умолчанию
```

```
1 def print_hello(language):  
2     match language:  
3         case "russian":  
4             print("Привет")  
5         case "english":  
6             print("Hello")  
7         case "german":  
8             print("Hallo")  
9  
10  
11 print_hello("english")    # Hello  
12 print_hello("german")    # Hallo  
13 print_hello("russian")   # Привет
```

## Задача блока 2

### ► НАУЧИТЬСЯ ГРАМОТНО ПИСАТЬ УСЛОВИЯ

```
if x <> y not 10 <= x, y > 7 elif  
print(true)
```

