# RUST
# Типы данных

# Введение

Rust является *статически типизированным* (statically typed) языком

## Скалярные типы данных

- целочисленный
- числа с плавающей точкой
- логический
- символы

## Составные типы данных

- кортежи
- массивы

# Целочисленный тип

- по умолчанию i32
- допускают использование суффикса типа, например 57u8
- могут использовать “_” в качестве визуального разделителя для облегчения чтения числа, например 1_000

| Числовой литерал | Пример |
|---|---|
| Десятичный | `98_222` |
| Шестнадцатеричный | `0xff` |
| Восьмеричный | `0o77` |
| Двоичный | `0b1111_0000` |
| Байт (только `u8`) | `b'A'` |

| Длина | Со знаком | Без знака |
|---|---|---|
| 8-бит | `i8` | `u8` |
| 16 бит | `i16` | `u16` |
| 32 бита | `i32` | `u32` |
| 64 бита | `i64` | `u64` |
| 128 бит | `i128` | `u128` |
| архитектурно-зависимая | `isize` | `usize` |

# Дробные числа

f32 - 32 бита

f64 - 64 бита (по умолчанию)

```rust
fn main() {
    let x = 2.0; // f64

    let y: f32 = 3.0; // f32
}
```

Все основные арифметические операции поддерживаются в **Rust**.

```rust
fn main() {
    // addition
    let sum = 5 + 10;

    // subtraction
    let difference = 95.5 - 4.3;

    // multiplication
    let product = 4 * 30;

    // division
    let quotient = 56.7 / 32.2;
    let truncated = -5 / 3; // Results in -1

    // remainder
    let remainder = 43 % 5;
}
```

# Логический тип

Имеет 2 возможных значения: ***true*** и ***false***. Занимают 1 байт.

```
fn main() {
    let t = true;

    let f: bool = false; // with explicit type annotation
}
```

# Символьный тип

Самый примитивный алфавитный тип данных.

```
fn main() {
    let c = 'z';
    let z: char = 'Z'; // with explicit type annotation
    let heart_eyed_cat = '😻';
}
```

Имеет размер 4 байта и является скалярным значением ***Unicode***.

!Важно! *Понятие “символа” в **Rust** нет!*

# Символьный тип

Рассмотрим строковый тип данных

```
let hello = "Здравствуйте";
let answer = &hello[0];
```

Все дело в том, как ***Rust*** хранит строки в памяти. Каждый символ русского языка занимает 8 байт **UTF-8**.

# Составные типы данных. Кортежи.

Кортеж- это универсальный способ объединения нескольких значений с различными типами в один составной тип.

- Имеют фиксированную длину: после объявления они не могут увеличиваться или уменьшаться в размерах
- Могут содержать разные типы
- Создаются с помощью круглых скобок ()
- Могут быть использованы как аргументы функции и как возвращаемые значения

# Кортежи

**Объявление:**

```
let tup: (i32, f64, u8) = (500, 6.4, 1);
```

**Получение отдельного значения:**

1 способ. Деструктуризация

```
let tup = (500, 6.4, 1);

let (x, y, z) = tup;

println!("The value of y is: {y}");
```

2 способ. Получение элемента по индексу

```
let x: (i32, f64, u8) = (500, 6.4, 1);

let five_hundred = x.0;

let six_point_four = x.1;
```

# Кортежи

Могут использоваться в качестве передаваемого аргумента в функции и в качестве возвращаемого значения.

Пример:

```
fn reverse(pair: (i32, bool)) -> (bool, i32) {
    // `let` можно использовать для создания связи
между кортежем и переменной
    let (integer, boolean) = pair;
    (boolean, integer)
}
```

# Кортежи

Кортежи могут содержать в себе кортежи

```
let tuple_of_tuples = ((1u8, 2u16, 2u32), (4u64, -1i8), -2i16);
```

**Печать кортежей**

1. Кортеж из одного элемента

```
println!("one element tuple: {:?}", (5u32,));
```

2. До 12 элементов

```
let pair = (1, true);
println!("pair is {:?}", pair);
```

3. Слишком длинный кортеж. Ошибка. Error[E0277]

```
let long = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12);
println!("{:?}", long);
```

# Массивы (Arrays)

Массив - набор элементов, при этом все элементы набора должны представлять один и тот же тип данных.

- каждый элемент массива должен иметь один и тот же тип
- имеют фиксированную длину
- удобно использовать, если данные необходимо разместить в стеке, а не в куче

# Массивы

1. Инициализация переменной массива

```
let переменная_массива: [тип_данных; размер] = [элемент1, элемент2, ...
let numbers: [i32; 7] = [1, 2, 3, 5, 8, 13, 21];
```

2. Объявление массива, который имеет 7 элементов типа **i32**

```
let название_массива: [тип_данных; размер];
let numbers: [i32; 7];
```

3. Заполнение массива значениями по умолчанию

```
let numbers: [i32; 5] = [2; 5];
```

4. Чтобы изменять элементы массива, нужен модификатор **mut**

```
let mut users = ["Tom", "Bob", "Sam"];
users[1] = "Bill";
```

# Массивы

Обращение к элементам массива

```
let users = ["Tom", "Bob", "Sam"];
println!("{}", users[0]);   //  Tom
println!("{}", users[2]);   //  Sam
```

При попытке обращения к несуществующему элементу мы столкнемся с ошибкой на этапе компиляции

```
let mut users = ["Tom", "Bob", "Sam"];

users[6] = "Bill";  // !Ошибка - элемента с индексом 6 в массиве users нет
```

# Массивы

Для перебора массива применяется цикл **for**:

```
let users = ["Tom", "Bob", "Sam"];
for user in users {
    print!("{} ", user);
}

println!(); // переходим на следующую строку в консоли

let numbers = [1, 2, 3, 5, 8, 13, 21];
for n in numbers{
    print!("{} ", n);
}
```