Rust

Pobieranie

https://www.rust-lang.org/tools/install (win)

https://doc.rust-lang.org/beta/book/ch01-01-installation.html
(caly docs z wszystkimi informacjami)

Podstawowe komendy

cargo new nazwa_projektu (Tworzenie pustego projektu)

cargo build (Kompliowanie)

cargo run (Kompilowanie + włączenie exe od razu)

cargo check (Sprawdzenie poprawności kodu bez konieczności całej kompilacji - krótsze)

cargo fmt (formatownie wszystkich plikow w projekcie)

Zmienne

Deklarujemy słowem **let.**
(np. **let** nazwa = 5;)

Opcjonalnie możemy określić typ
(np. let nazwa**: i32** = 100;)

Zmienną możemy określić jako modyfikowalną słowem **mut** (od mutable).
(np. let mut zmienna = 5;).

Printy

- **print!("tekst") - wyświetla tekst**
- **println!("tekst") - wyświetla tekst i dodaje na końcu nowa linijkę**

Przykład ze zmienna:

```
println!("liczba1 = {},  liczba2 = {}", x, y)
```

Typy całkowitoliczbowe (int)

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Liczby zmiennoprzecinkowe (float)

- Mamy dwa typy zmiennoprzecinkowe:
  -f32
  -f64

- Tak samo jak w intach liczba to liczba bitow na jakiej zapisujemy dane

Pozostałe podstawowe typy

- char - może przechowywać każda wartość unicode – 32 bity

  Przykład - let x: char = '中';

- bool – przechowuje true or false – 1 bit

  Przyklad – let y: bool = true;

- String – tekst – później będzie rozwinięte

  Przyklad – let text: String = String::from("jakis tam tekst")

Konwersja typów liczbowych

- Rust wymaga jawnych konwersji danych
- Zawsze gdy konwertujemy z mniejszego na wiekszy typ musimy uważać na utrate danych

Pętle

- Mamy 3 rodzaje pętli:
  standardowe while i for,
  oraz nieskończoną pętle loop którą
  musimy zakończyć breakiem
  wewnątrz jej

Funkcje

- **`Słowo fn`** do definiowania funkcji
- Nazwa
- Parametry w nawiasach `()`, `obowiązkowo określony typ`
- Typ zwracany po strzałce `->`
- `Cialo funkcji w klamrach`

- `Return zwraca wartosc`

Ownership

- **Każda wartość ma swojego właściciela**
- Jedna zmienna = jeden właściciel

- **Zasada przenoszenia (move)**
- Gdy przypiszemy zmienną do innej zmiennej, własność jest przenoszona
- Stara zmienna przestaje być dostępna

- **Zasada zakresu (scope)**
- Gdy właściciel wychodzi poza zakres, wartość jest automatycznie zwalniana

Borrowing

- Możesz mieć dowolną liczbę referencji współdzielonych (&T) **LUB** jedną referencję mutowalną (&mut T) - nigdy obu naraz.
- Referencje zawsze muszą być prawidłowe - nie mogą wskazywać na usunięte dane.
- Właściciel danych nie może modyfikować wartości, dopóki istnieją do niej referencje.
- Referencja nie może przeżyć dłużej niż dane, do których się odwołuje.

Ownership i Borrowing w funkcjach

- **Przekazanie wartości do funkcji:**
- Albo funkcja ZABIERA własność (wartość przepada)
- Albo funkcja POŻYCZA wartość (& = tylko patrzy)
- Albo funkcja MODYFIKUJE wartość (&mut = może zmieniać)
- **Zasady:**
- Co ZABRANE, tego nie użyjemy ponownie
- Co POŻYCZONE, możemy nadal używać
- Co ZMODYFIKOWANE, zmienia się na stałe

String i &str

- String
  - Modyfikowalny
  - String przechowuje adres w pamieci tekstu, dlugosc tekstu, i pojemnosc
  - Jest wlascicielem danych
- &str
  - Niemodyfikowalny
  - Przechowuje adres w pamieci tekstu i jego dlugosc
  - Nie jest wlascicielem danych, tylko je pożycza