

## Лабораторна робота 1. Знайомство з RUST.

### 1. Встановлення Rust

Завантажимо Rust за допомогою rustup, інструмента командного рядка для керування виданнями Rust і пов'язаних інструментів.

Встановлення rustup на Windows

Перейдіть до <https://www.rust-lang.org/tools/install> і дотримуйтеся вказаних там інструкцій для встановлення Rust. У певний момент встановлення ви отримаєте повідомлення, що вам також знадобляться інструменти збірки MSVC для Visual Studio 2013 чи пізнішої. Щоб отримати інструменти збірки, вам потрібно встановити Visual Studio 2022.

```
C:\Users\alyon\Downloads\ru x + v

Rust Visual C++ prerequisites

Rust requires a linker and Windows API libraries but they don't seem to be available.

These components can be acquired through a Visual Studio installer.

1) Quick install via the Visual Studio Community installer
   (free for individuals, academic uses, and open source).

2) Manually install the prerequisites
   (for enterprise and advanced users).

3) Don't install the prerequisites
   (if you're targeting the GNU ABI).

>1

info: downloading Visual Studio installer
info: running the Visual Studio install
info: rustup will continue once Visual Studio installation is complete
```

```
C:\Users\alyon\Downloads\ru x + v

info: downloading component 'rustfmt'
info: installing component 'cargo'
info: installing component 'clippy'
info: installing component 'rust-docs'
14.3 MiB / 14.3 MiB (100 %) 457.6 KiB/s in 3m 13s ETA: 0s
info: installing component 'rust-std'
17.9 MiB / 17.9 MiB (100 %) 12.8 MiB/s in 1s ETA: 0s
info: installing component 'rustc'
58.7 MiB / 58.7 MiB (100 %) 13.2 MiB/s in 4s ETA: 0s
info: installing component 'rustfmt'
info: default toolchain set to 'stable-x86_64-pc-windows-msvc'

stable-x86_64-pc-windows-msvc installed - rustc 1.75.0 (82e1608df 2023-12-21)

Rust is installed now. Great!

To get started you may need to restart your current shell.
This would reload its PATH environment variable to include
Cargo's bin directory (%USERPROFILE%\cargo\bin).

Press the Enter key to continue.
```

### 2. Перший запуск. Hello, World!

Після встановлення Rust напишемо першу програму цією мовою.

#### 2.1 Створення теки проєкту

Для початку, створіть теку для розміщення вашого коду мовою Rust.

```
> mkdir "%USERPROFILE%\projects"
> cd /d "%USERPROFILE%\projects"
> mkdir hello_world
> cd hello_world
```

## **2.2 Написання і запуск програми на Rust**

Тепер створіть новий вихідний файл і назвіть його main.rs. Файли Rust завжди закінчуються розширенням .rs. Якщо у назві файлу використовується більш ніж одне слово, домовлено для розділення використовувати підкреслення.

Тепер відкрийте файл main.rs, який ви щойно створили, і наберіть код:

```
fn main() {
    println!("Hello, world!");
}
```

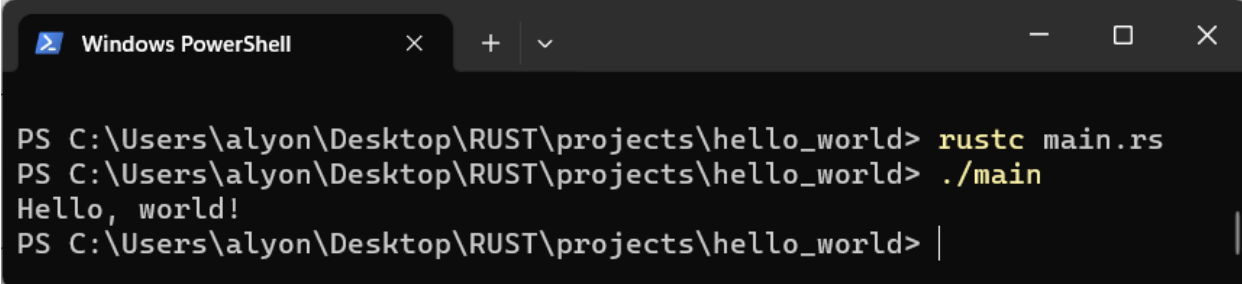
Збережіть цей файл і поверніться до вікна термінала у теці ~/projects/hello\_world.

Запустіть команду .\main.exe:

```
rustc main.rs
```

```
.\main.exe
```

```
Hello, world!
```



```
Windows PowerShell
PS C:\Users\alyon\Desktop\RUST\projects\hello_world> rustc main.rs
PS C:\Users\alyon\Desktop\RUST\projects\hello_world> ./main
Hello, world!
PS C:\Users\alyon\Desktop\RUST\projects\hello_world> |
```

Якщо вивелося Hello, world! - вітаємо! Ви щойно офіційно написали програму мовою Rust.

## **3. Cargo**

Cargo - це система побудови та пакетний менеджер Rust.

Перевірити, чи встановлений Cargo можна, ввівши це у свій термінал:

```
cargo --version
```

Якщо ви побачите номер версії, то Cargo встановлений!

### **3.1 Створення проєкту за допомогою Cargo**

Створімо новий проєкт за допомогою Cargo і подивімося, як він відрізняється від нашого початкового проєкту Hello World.

```
cargo new hello_cargo
```

```
cd hello_cargo
```

Перша команда створює нову теку і проєкт, що зветься hello\_cargo. Ми назвали наш проєкт hello\_cargo, і Cargo створює свої файли у теці з такою назвою.

Перейдіть до теки hello\_cargo і перегляньте файли. Ви побачите, що Cargo створив два файли і одну теку: Cargo.toml і теку src із файлом main.rs.

Також він розпочав новий репозиторій Git, додавши файл .gitignore. Файли Git не будуть створені, якщо ви запустите cargo new в уже створеному репозиторії Git; ви можете змінити цю поведінку за допомогою cargo new --vcs=git.

Відкрийте файл Cargo.toml у будь-якому текстовому редакторі.

```
[[package]
name = "hello_cargo"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
```

Це файл у форматі TOML (Tom's Obvious, Minimal Language - "Томова очевидна мінімальна мова"), який Cargo використовує як формат для конфігурації.

Перший рядок, [package] (пакет) - це заголовок розділу, що показує, що наступні інструкції стосуються конфігурації пакета. Коли ми додамо більше інформації до цього файлу, ми додамо й інші розділи.

Наступні три рядки встановлюють конфігураційну інформацію, потрібну Cargo для компілювання вашої програми: ім'я, версію і яке видання Rust використовувати.

Останній рядок, [dependencies], розпочинає розділ, де можна вказувати залежності вашого проєкту.

Тепер відкрийте файл src/main.rs і подивіться на його вміст:

```
fn main() {
    println!("Hello, world!");
}
```

Cargo створив для вас "Hello World!" Поки що відмінності між нашим попереднім проєктом та згенерованим Cargo полягає в тому, що Cargo розмістив код у теці src і додав конфігураційний файл Cargo.toml в основній теці.

Cargo очікує, що вихідні файли будуть розташовані в теці src, а основна тека міститиме лише README, ліцензійну інформацію, конфігураційні файли й інше.

### ***3.2 Побудова і запуск проєкту Cargo***

Погляньмо, як відрізняється збірка і запуск програми "Hello, world!" за допомогою Cargo. Зберіть проєкт такими командами з теки hello\_cargo:

cargo build

```
Compiling hello_cargo v0.1.0 (C:\Users\alyon\Desktop\RUST\projects\hello_cargo)
Finished dev [unoptimized + debuginfo] target(s) in 1.90s
```

Ця команда створить виконаний файл. Cargo розміщує двійковий файл у теці, що зветься debug. Виконуваний файл можна запустити такою командою:

./target/debug/hello\_cargo

```
Hello, world!
```

Якщо все пройшло добре, в термінал виведеться Hello, world!

Використання `cargo run` зручніше, ніж запам'ятовувати виконати `cargo build`, а потім писати весь шлях до двійкового файлу, тому більшість розробників використовують `cargo run`.

```
cargo run
```

```
Finished dev [unoptimized + debuginfo] target(s) in 0.04s
Running `target\debug\hello_cargo.exe`
Hello, world!
```

Зверніть увагу, що цього разу ми не побачили повідомлення про те, що Cargo компілює `hello_cargo`. Cargo зрозумів, що файли не змінилися, тому не перезібрав, а просто запустив двійковий файл. Якби ви змінили вихідний код, Cargo б довелося перебудувати проєкт перед виконанням, і ви б побачили такий вивід:

```
cargo run
```

```
Compiling hello_cargo v0.1.0 (C:\Users\alyon\Desktop\RUST\projects\hello_cargo)
Finished dev [unoptimized + debuginfo] target(s) in 0.51s
Running `target\debug\hello_cargo.exe`
Hello, world!
```

Крім того, Cargo має команду `cargo check`. Ця команда швидко перевіряє ваш код, щоб переконатися, що він компілюється, але не створює виконанного файлу:

```
cargo check
```

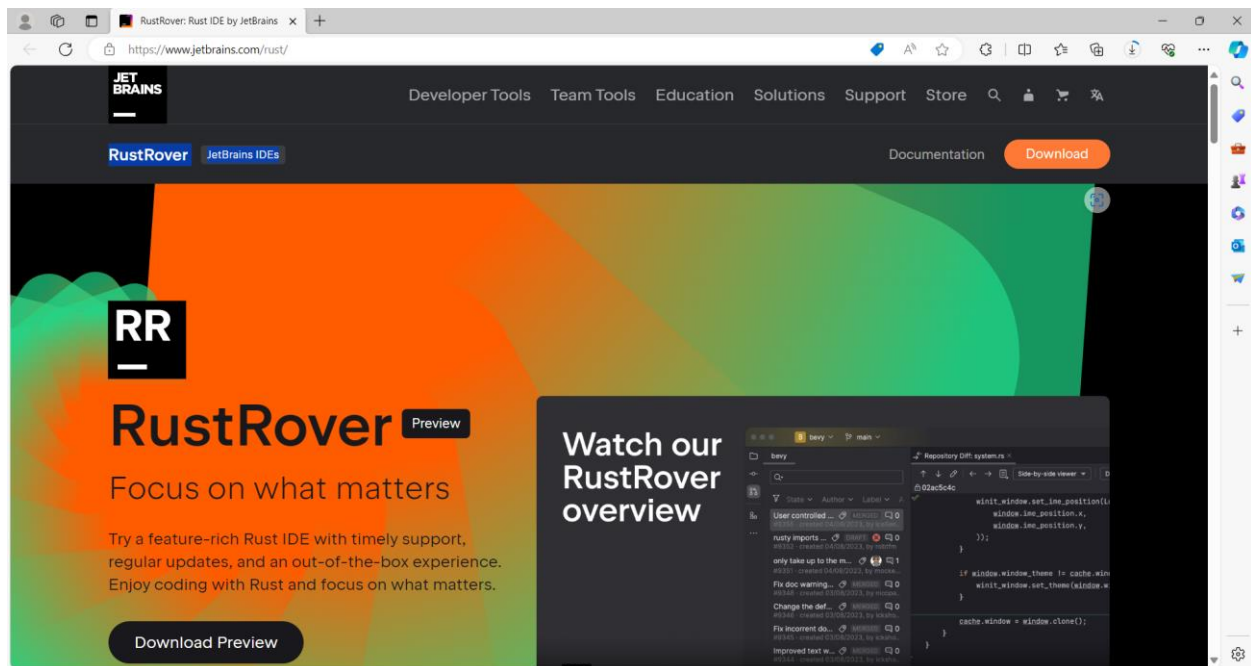
Підіб'ємо підсумок, що ж ми дізналися про Cargo:

- ✓ Ми можемо створити проєкт за допомогою `cargo new`.
- ✓ Ми можемо зібрати проєкт за допомогою `cargo build`.
- ✓ Ми можемо зібрати і запустити проєкт в одну дію за допомогою `cargo run`.
- ✓ Ми можемо зібрати проєкт без створення двійкового файлу для пошуку помилок за допомогою `cargo check`.
- ✓ Cargo зберігає результат збірки не в одній теці з кодом, а в теці `target/debug`.

Додаткова перевага використання Cargo полягає в тому, що його команди однакові незалежно від операційної системи, в якій ви працюєте.

#### 4. Завантаження та встановлення IDEs RustRover

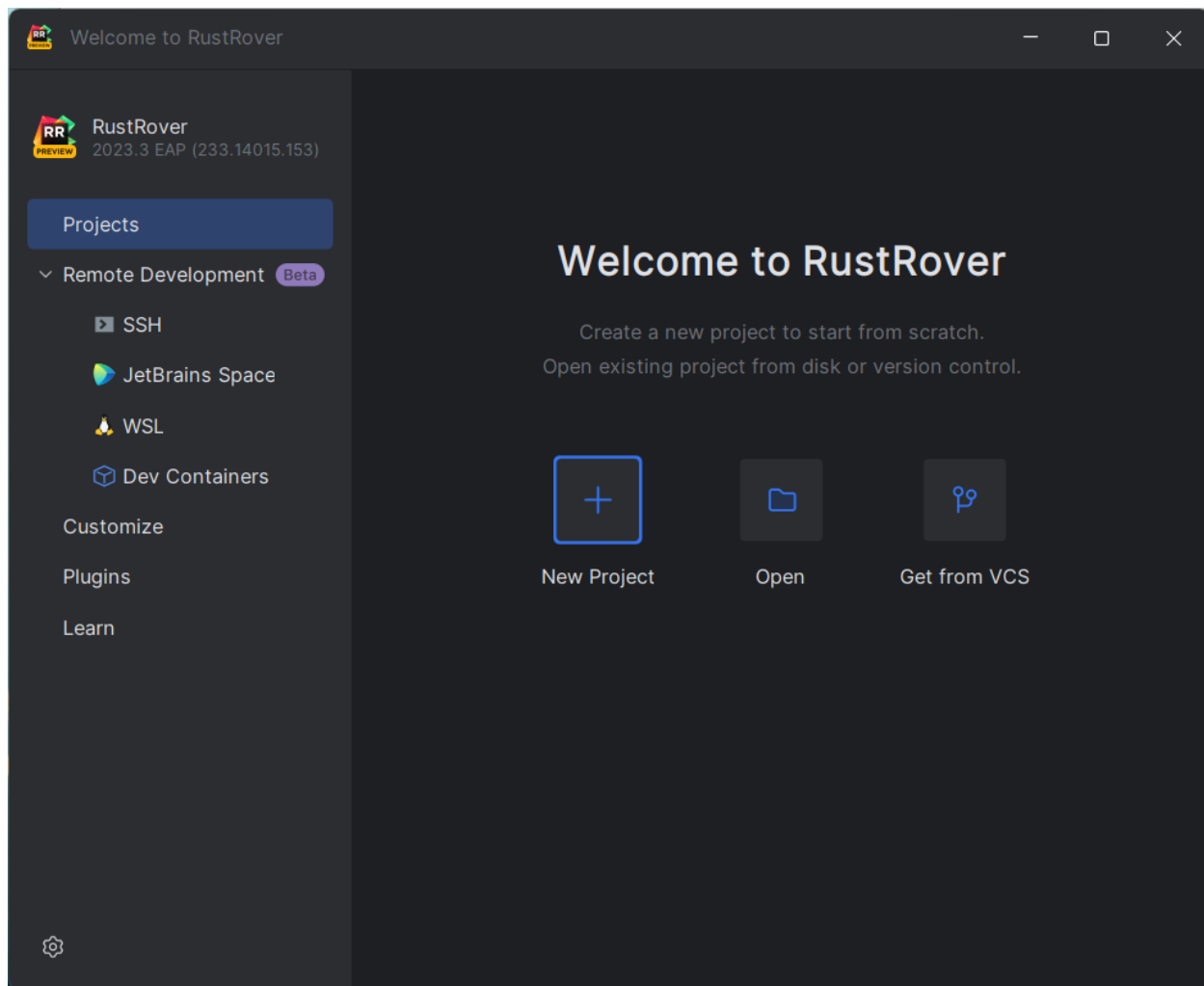
Для створення програм на RUST будемо використовувати безкоштовне середовище розробки від JetBrains IDEs – RustRover, яке можна завантажити за наступною адресою: <https://www.jetbrains.com/rust/>



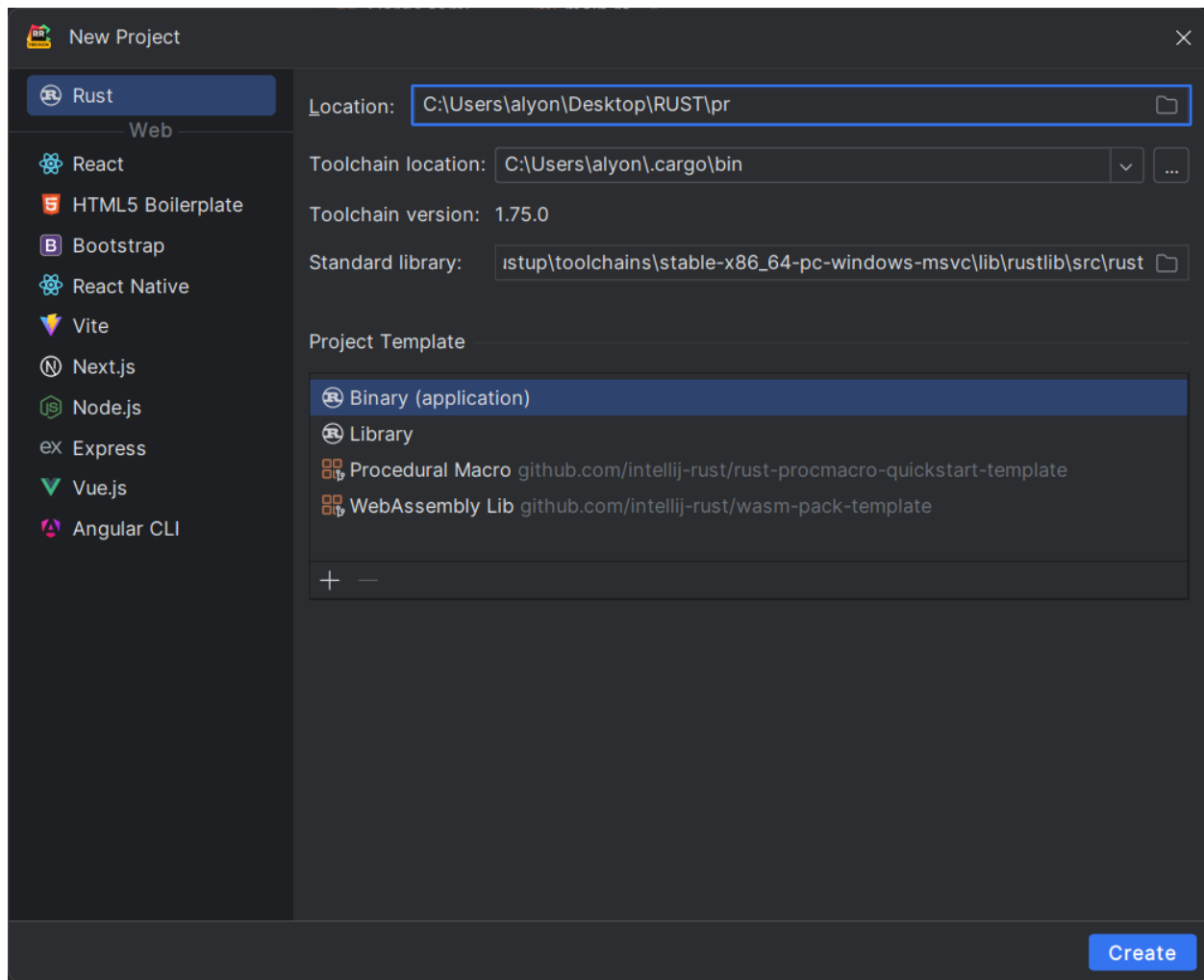
Після завантаження запускаємо програму інсталятора та виконуємо встановлення.

#### ***4.1. Створення проекту***

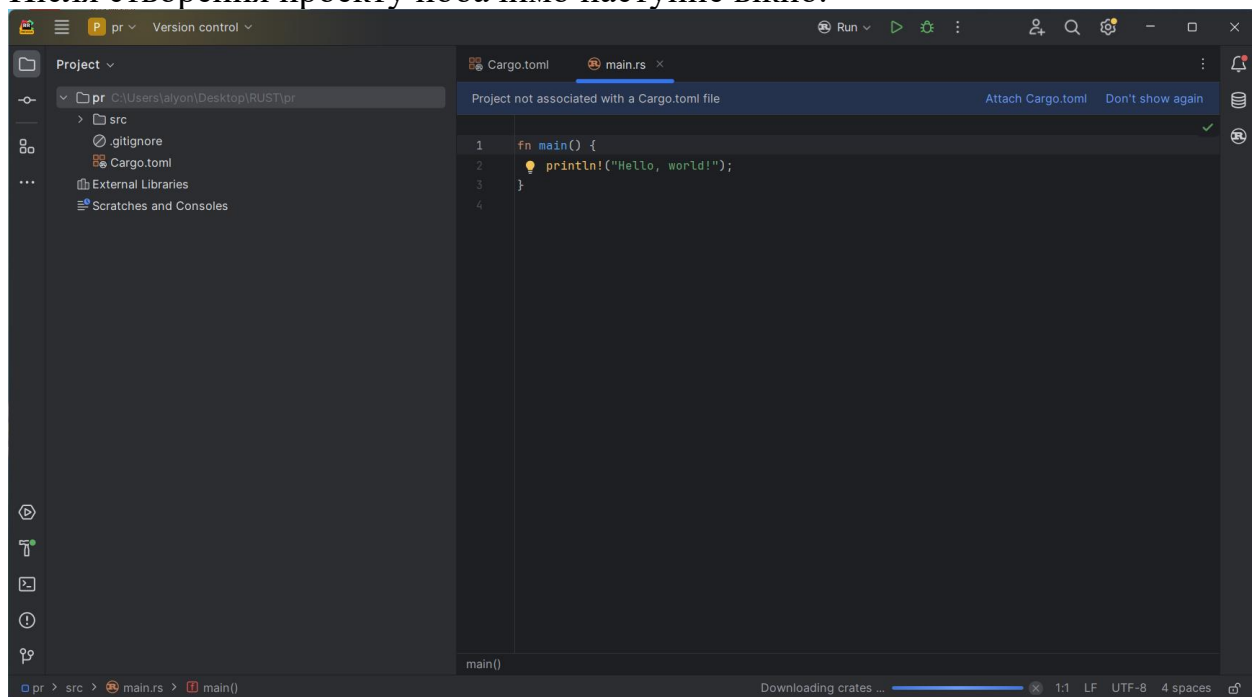
Запускаємо RustRover. На стартовому екрані обираємо Create a new project (Створити новий проект)



Наступним кроком необхідно вказати ім'я та каталог для розміщення проекту, натискаємо Create (Створити).



Після створення проекту побачимо наступне вікно.



Структура проекту знаходиться ліворуч, в якому можна побачити структуру нашого проекту.

## 5. Завдання

5.1 Ряд Фібоначчі. Створіть функцію `fib`, яка приймає параметр `n` і повертає список перших `n` чисел ряду Фібоначчі. Наприклад:

Для `n = 1`, функція повинна повернути список `[0, 1, 1]`.

Для `n = 2`, функція повинна повернути список `[0, 1, 1, 2]`.

Для `n = 5`, функція повинна повернути список `[0, 1, 1, 2, 3, 5]`.

```
fn fib(n: u32) -> u32 {
    if n <= 2 {
        // The base case.
    } else {
        // The recursive case.
    }
}

fn main() {
    let n = 20;
    println!("fib(n) = {}", fib(n));
}
```

5.2 Напишіть програму, яка виводить на екран квадрат Піфагора - таблицю множення. Рекомендований вигляд екрану:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90