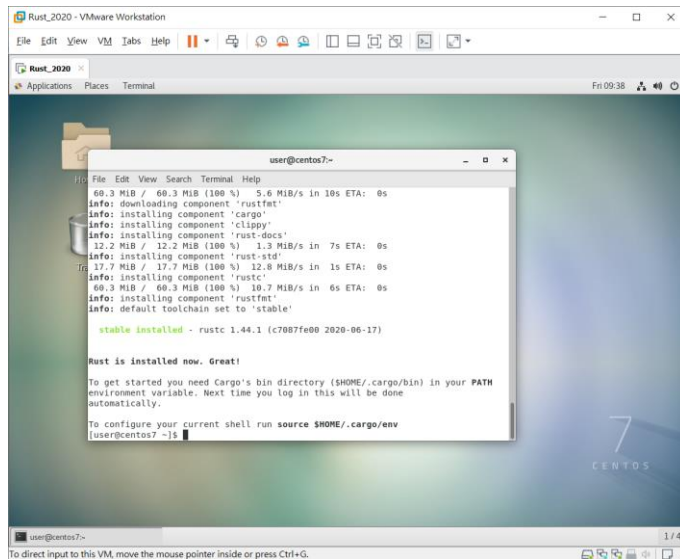


# Rust 學習歷程

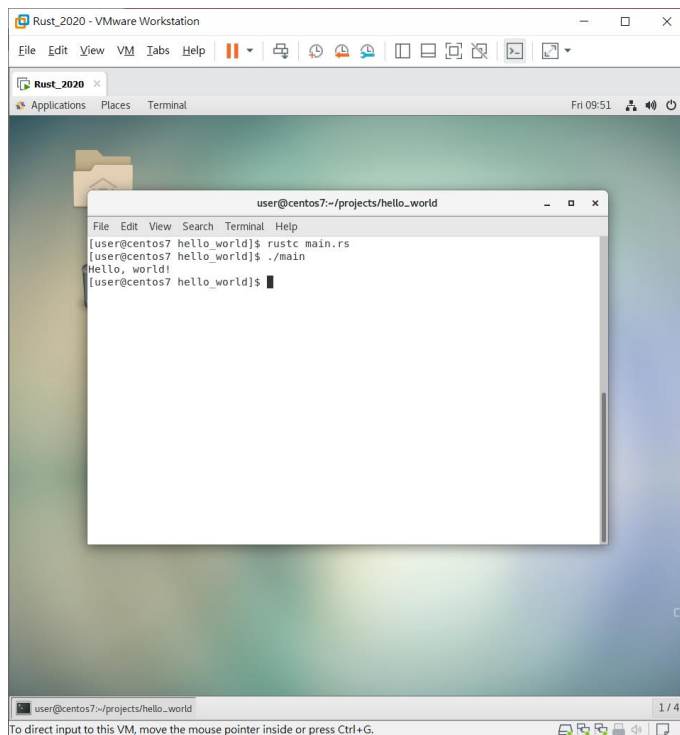
## -安裝



使用以下指令下載 rustup

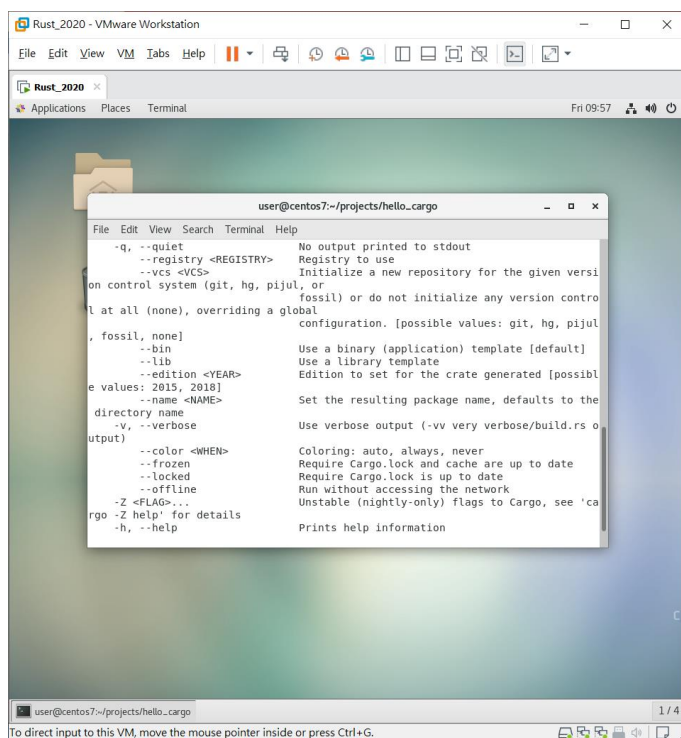
```
$ curl https://sh.rustup.rs -sSf | sh
```

運行簡單的小程式



運行的程式要透過 rustc 另外生成

## cargo 可用的指令



## cargo 程式說明

```
[package]           //是一個片段標題
name = "hello_cargo"    //配置項目的名稱
version = "0.1.0"       //配置項目的版本
authors = ["Your Name <you@example.com>"] //配置項目的作者
edition = "2018"
```

```
[dependencies] //列出項目依賴的片段的開始
```

cargo 目的說明:

Cargo 期望原始檔案存放在 `src` 目錄中。專案根目錄只存放 `README`、`license` 資訊、設定檔和其他跟代碼無關的檔。使用 `Cargo` 幫助你保持項目乾淨整潔，一切井井有條。

## 重點整理

```
println!("Hello, world!"); //印出想要的字串內容
$ cargo build //建構 cargo
$ cargo run //運行 cargo
$ cargo check //快速檢查代碼確保其可以編譯，但並不產生可執行文件
$ cargo update //更新 cargo 到最新版本
```

## -練習編寫一個猜數字遊戲

程式說明:

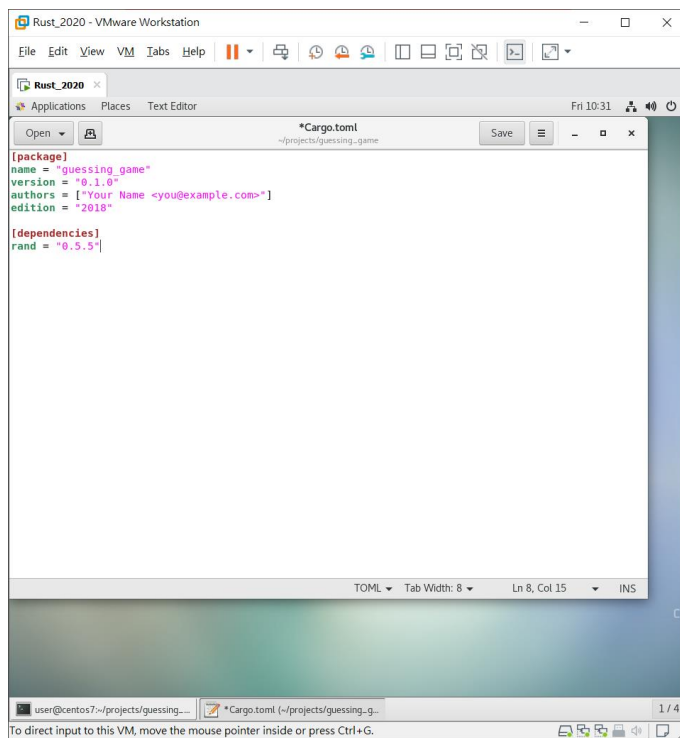
```
use std::io; //引入標準輸入輸出
fn main() { //程式入口
    println!("Guess the number!"); //印出 Guess the number!

    println!("Please input your guess.");

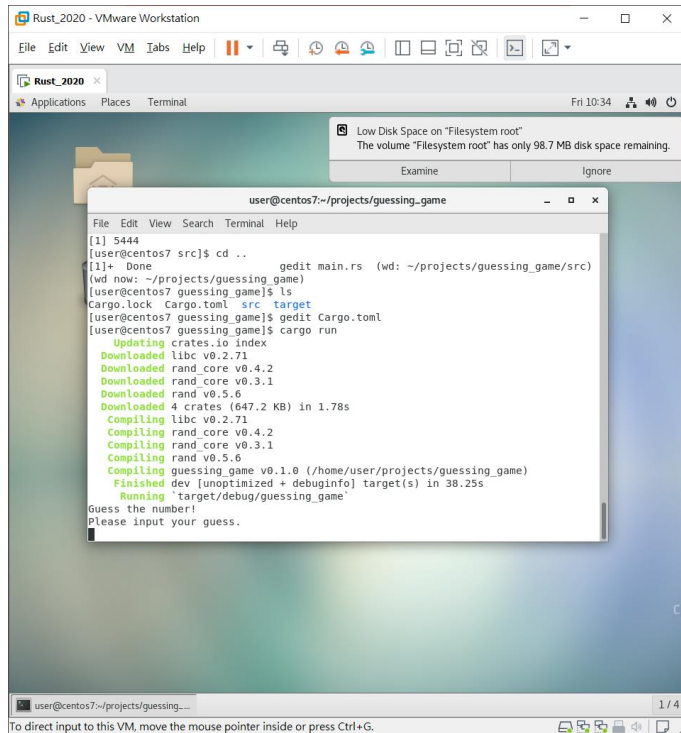
    let mut guess = String::new(); //創建一個儲存用戶輸入的地方

    io::stdin().read_line(&mut guess) //無論鍵入甚麼，都存入一個字串
        .expect("Failed to read line"); //除錯

    println!("You guessed: {}", guess); //使用格式化字串後的第一個
    值
}
```



將 rand 加入代碼包



程式說明:產生了一個隨機變數並印出，保留輸入數字功能

```
use std::io;

use rand::Rng; //定義隨機生成器所使用函式庫

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}", secret_number);

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {}", guess);
}
```

程式說明:加入 loop 的程式碼

```
use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1, 101);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

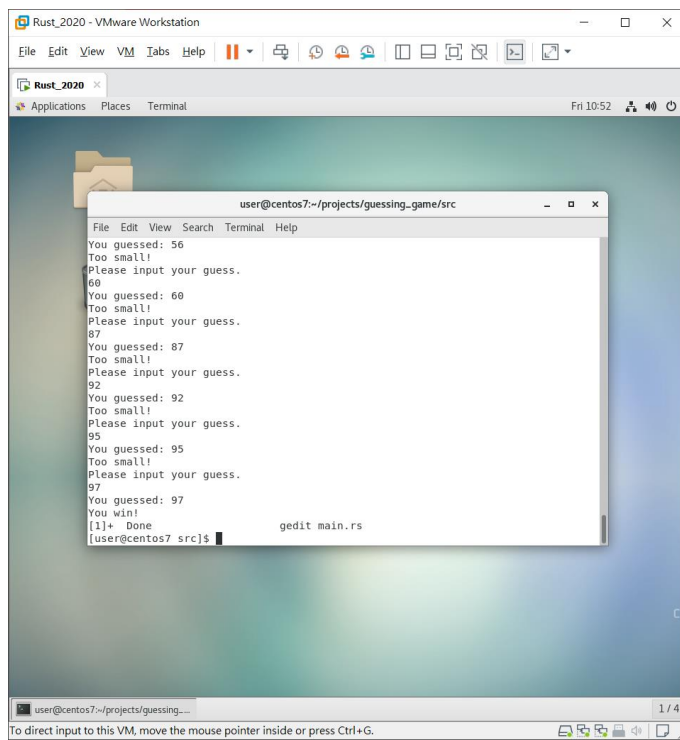
        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

執行結果：



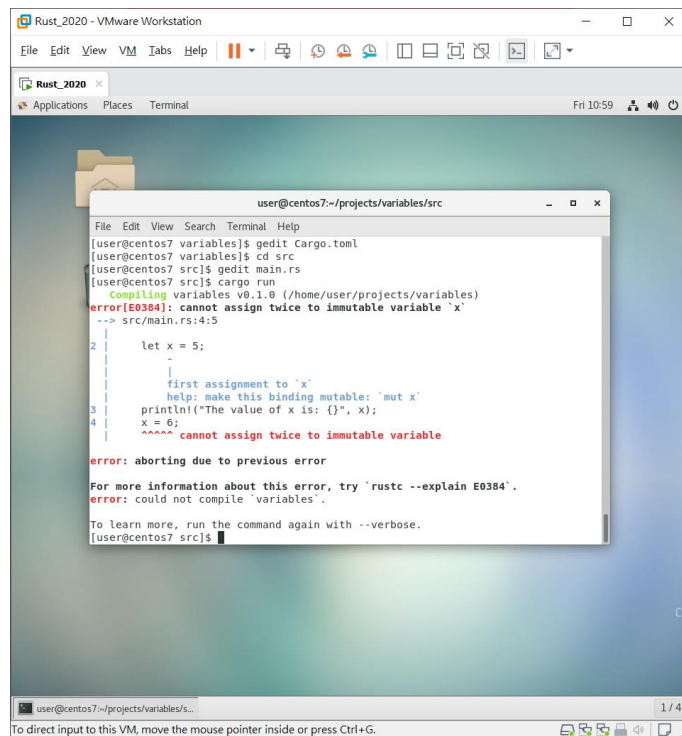
```
user@centos7:~/projects/guessing_game/src
File Edit View Search Terminal Help
You guessed: 56
Too small!
Please input your guess.
60
You guessed: 60
Too small!
Please input your guess.
87
You guessed: 87
Too small!
Please input your guess.
92
You guessed: 92
Too small!
Please input your guess.
95
You guessed: 95
Too small!
Please input your guess.
97
You guessed: 97
You win!
[1]+  Done                  gedit main.rs
[user@centos7 src]$
```

## -變量及可變性:

當變數不可變時，一旦值被綁定一個名稱上，你就不能改變這個值。為了對此進行說明，使用 `cargo new variables` 命令在 `projects` 目錄生成一個叫做 `variables` 的新項目。

程式錯誤說明:

錯誤資訊指出錯誤的原因是 不能對不可變變數 `x` 二次賦值（`cannot assign twice to immutable variable x`），因為你嘗試對不可變變數 `x` 賦第二個值。

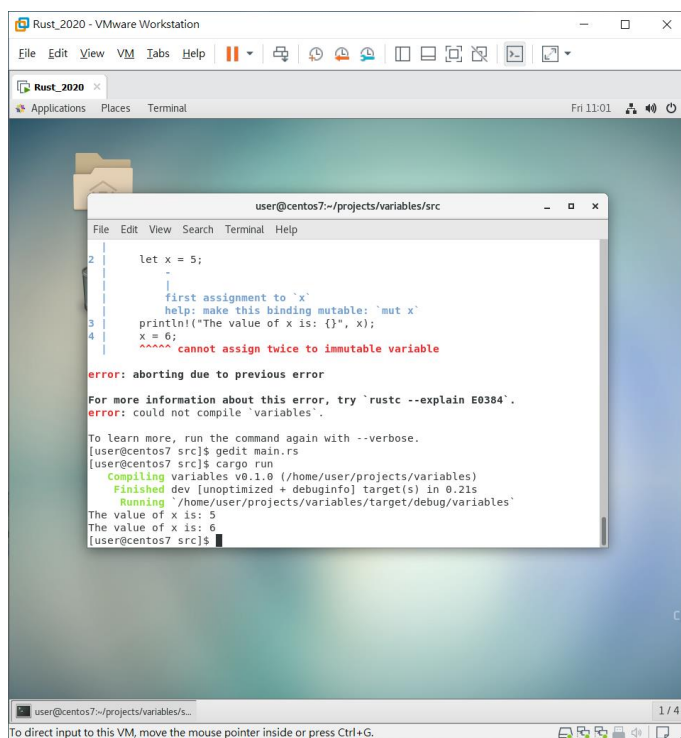


## 修改方式:

```
fn main() {  
    let mut x = 5;  
    println!("The value of x is: {}", x);  
    x = 6;  
    println!("The value of x is: {}", x);  
}
```

通過 `mut`，允許把綁定到 `x` 的值從 5 改成 6。在一些情況下，你會想用可變變數，因為與只用不可變變數相比，它會讓代碼更容易編寫。

更改後執行：



The screenshot shows a Rust 2020 VM window with a terminal open. The terminal displays the following code and output:

```
2 | let x = 5;
   |
   | first assignment to 'x'
   | help: make this binding mutable: 'mut x'
3 | println!("The value of x is: {}", x);
   |
4 | x = 6;
   | ^^^^^ cannot assign twice to immutable variable

error: aborting due to previous error

For more information about this error, try `rustc --explain E0384`.
error: could not compile `variables`.

To learn more, run the command again with --verbose.
[user@centos7 src]$ gedit main.rs
[user@centos7 src]$ cargo run
   Compiling variables v0.1.0 (/home/user/projects/variables)
  Finished dev [unoptimized + debuginfo] target(s) in 0.21s
   Running `/home/user/projects/variables/target/debug/variables'
The value of x is: 5
The value of x is: 6
[user@centos7 src]$
```

## 變量和常量的區別

不允許改變值的變數，可能會使你想起另一個大部分程式設計語言都有的概念：**常量**（*constants*）。類似不可變變數，常量是綁定到一個名稱的不允許改變的值，不過常量與變數還是有一些區別。

首先，不允許對常量使用 `mut`。常量不光默認不能變，它總是不能變。聲明常量使用 `const` 關鍵字而不是 `let`，並且 必須 注明值的類型。常量可以在任何作用域中聲明，包括全域作用域，這在一個值需要被很多部分的代碼用到時很有用。

最後一個區別是，常量只能被設置為常量運算式，而不能是函式呼叫的結果，或任何其他只能在運行時計算出的值。

將遍佈于應用程式中的硬編碼值聲明為常量，能幫助後來的代碼維護人員瞭解值的意圖。如果將來需要修改硬編碼值，也只需修改彙聚於一處的硬編碼值。

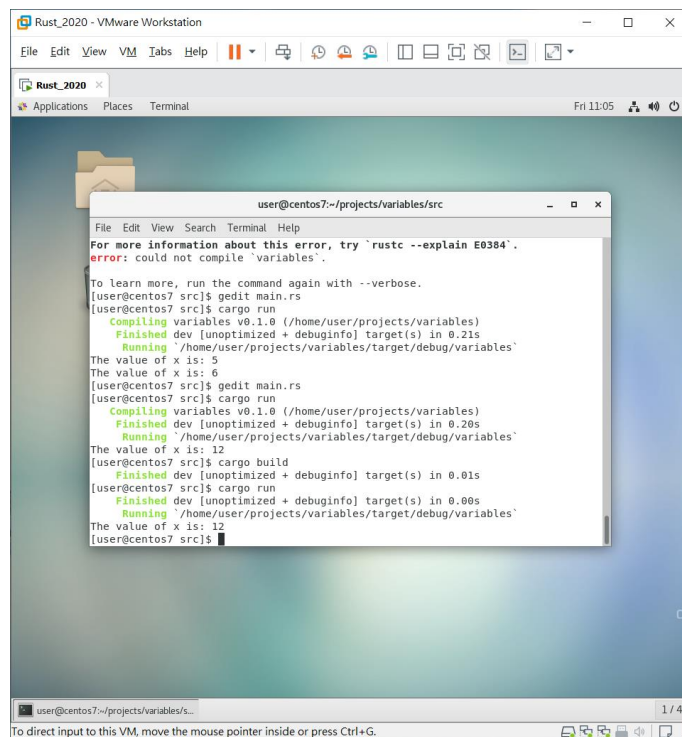


## -隱藏

程式說明：這個程式首先將 `x` 綁定到值 `5` 上。接著通過 `let x = 隱藏 x`，獲取初始值並加 `1`，這樣 `x` 的值就變成 `6` 了。第三個 `let` 語句也隱藏了 `x`，將之前的值乘以 `2`，`x` 最終的值是 `12`。

```
fn main() {  
    let x = 5;  
  
    let x = x + 1;  
  
    let x = x * 2;  
  
    println!("The value of x is: {}", x);  
}
```

運行結果：



The screenshot shows a terminal window titled "Rust\_2020 - VMware Workstation" with a sub-window titled "user@centos7:~/projects/variables/src". The terminal displays the output of a Rust program. It starts with a compilation error: "error: could not compile 'variables'". The user then runs the program with the command "cargo run", which outputs "The value of x is: 5". The user then edits the file with "gedit main.rs" and runs "cargo run" again, which outputs "The value of x is: 6". Finally, the user runs "cargo build" and "cargo run" again, which outputs "The value of x is: 12". The terminal window also shows the command "rustc --explain E0384" and the output "To learn more, run the command again with --verbose."

## -標量類型

整數型態：

**整數** 是一個沒有小數部分的數位。我們在第二章使用過 `u32` 整數類型。該型別宣告表明，它關聯的值應該是一個佔據 32 比特位元的不帶正負號的整數（有符號整數類型以 `i` 開頭而不是 `u`）。在有符號列和無符號列中的每一個變體（例如，`i16`）都可以用來聲明整數值的類型。

長度	有符號	無符號
8-bit	<code>i8</code>	<code>u8</code>
16-bit	<code>i16</code>	<code>u16</code>
32-bit	<code>i32</code>	<code>u32</code>
64-bit	<code>i64</code>	<code>u64</code>
128-bit	<code>i128</code>	<code>u128</code>
arch	<code>isize</code>	<code>usize</code>

每一個變體都可以是有符號或無符號的，並有一個明確的大小。**有符號** 和 **無符號** 代表數位能否為負值，換句話說，數位是否需要有一個符號（有符號數），或者永遠為正而不需要符號（無符號數）。這有點像在紙上書寫數字：當需要考慮符號的時候，數位以加號或減號作為首碼；然而，可以安全地假設為正數時，加號首碼通常省略。有符號數以補數形式儲存。

浮点型：

**Rust** 也有兩個原生的 浮點數（floating-point numbers）類型，它們是帶小數點的數位。**Rust** 的浮點數類型是 `f32` 和 `f64`，分別占 32 位和 64 位。默認類型是 `f64`，因為在現代 CPU 中，它與 `f32` 速度幾乎一樣，不過精度更高。

布林型

正如其他大部分程式設計語言一樣，**Rust** 中的布林類型有兩個可能的值：`true` 和 `false`。**Rust** 中的布林類型使用 `bool` 表示。

## -複合類型

### 元組類型

元組是一個將多個其他類型的值組合進一個複合類型的主要方式。元組長度固定：一旦聲明，其長度不會增大或縮小。

### 陣列類型

另一個包含多個值的方式是 陣列（**array**）。與元組不同，陣列中的每個元素的類型必須相同。**Rust** 中的陣列與一些其他語言中的陣列不同，因為 **Rust** 中的陣列是固定長度的：一旦聲明，它們的長度不能增長或縮小。

我們使用包含在圓括號中的逗號分隔的值列表來創建一個元組。元組中的每一個位置都有一個類型，而且這些不同值的類型也不必是相同的。

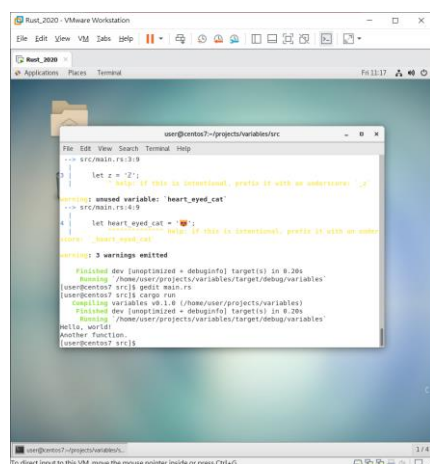
## -函式

### 普通函式

程式說明：

```
fn main() {  
    println!("Hello, world!");  
  
    another_function();  
}  
  
fn another_function() {  
    //創建一個 another_function 函式  
    println!("Another function.");  
}
```

執行結果：

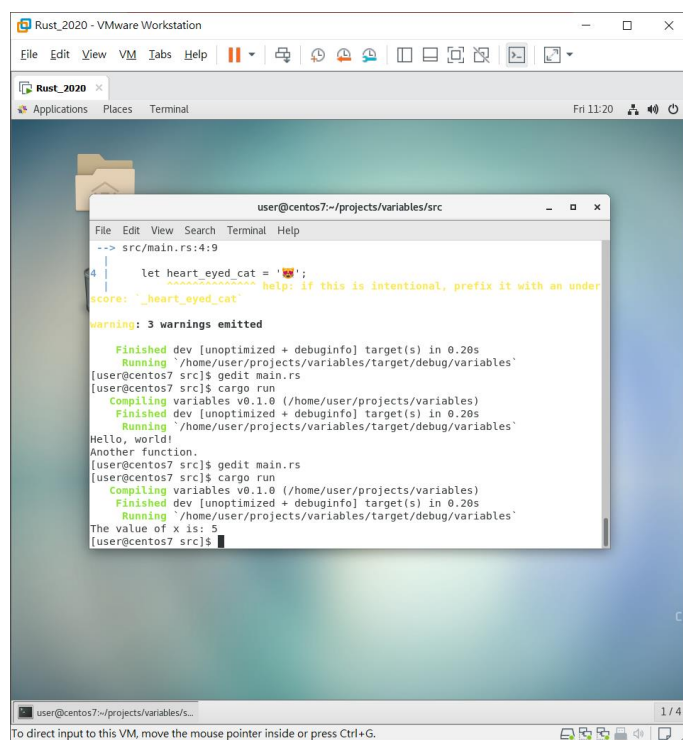


含有參數的函式

程式說明:

```
fn main() {  
    another_function(5);    將函式的 x 值帶入 5  
}  
  
fn another_function(x: i32) {  
    println!("The value of x is: {}", x);  
}
```

執行結果:

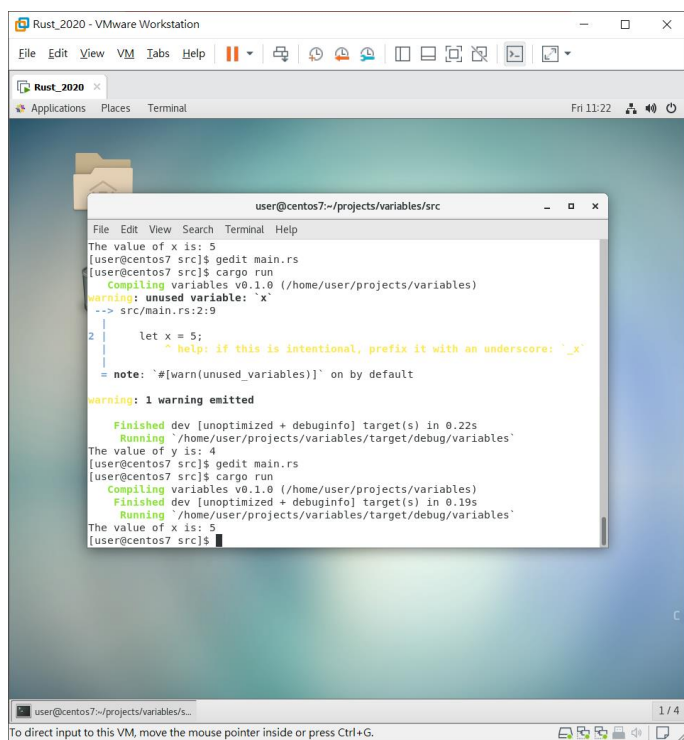


回傳值的函式

程式說明:

```
fn five() -> i32 {  
    5    //看起來不完整卻代表傳回 5 的值  
}  
  
fn main() {  
    let x = five();  
  
    println!("The value of x is: {}", x);  
}
```

執行結果:



```
user@centos7:~/projects/variables/src
File Edit View Search Terminal Help
The value of x is: 5
[user@centos7 src]$ gedit main.rs
[user@centos7 src]$ cargo run
Compiling variables v0.1.0 (/home/user/projects/variables)
warning: unused variable: `x`
--> src/main.rs:2:9
2 | let x = 5;
  |         ^ help: if this is intentional, prefix it with an underscore: `_x`
  |
note: `[warn(unused_variables)]` on by default
warning: 1 warning emitted
Finished dev [unoptimized + debuginfo] target(s) in 0.22s
Running `/home/user/projects/variables/target/debug/variables`
The value of y is: 4
[user@centos7 src]$ gedit main.rs
Compiling variables v0.1.0 (/home/user/projects/variables)
Finished dev [unoptimized + debuginfo] target(s) in 0.19s
Running `/home/user/projects/variables/target/debug/variables`
The value of x is: 5
[user@centos7 src]$
```

-註解

```
// hello, world
```

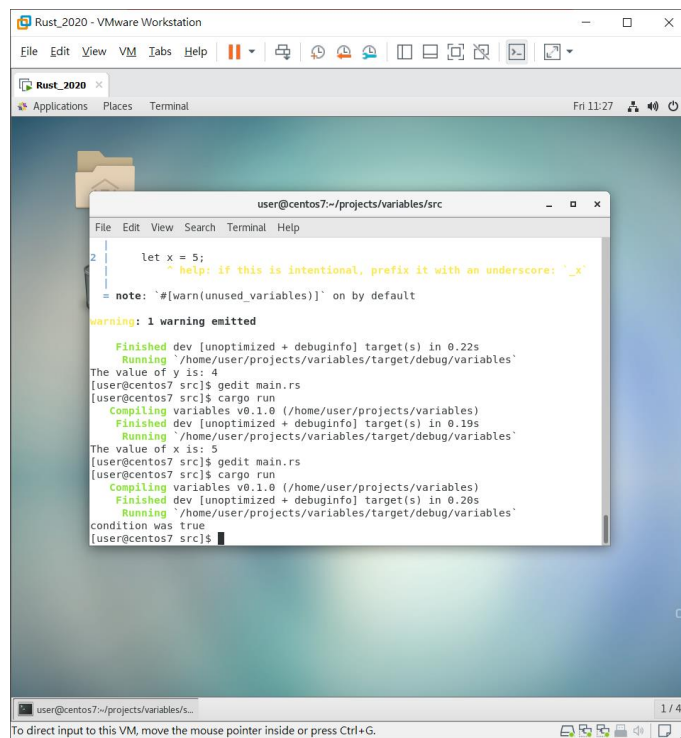
所有程式師都力求使其代碼易於理解，不過有時還需要提供額外的解釋。在這種情況下，程式師在源碼中留下記錄，或者 注釋（**comments**），編譯器會忽略它們，不過閱讀代碼的人可能覺得有用。

## -if 表達式

程式說明:

```
fn main() {  
    let number = 3;  
  
    if number < 5 {  
        println!("condition was true");  
    } else {  
        println!("condition was false");  
    }  
}
```

執行結果:



The screenshot shows a terminal window titled "user@centos7:~/projects/variables/src" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal output shows the execution of a Rust program. It starts with a warning about unused variables, followed by the compilation and execution of the program. The output of the program is "condition was true".

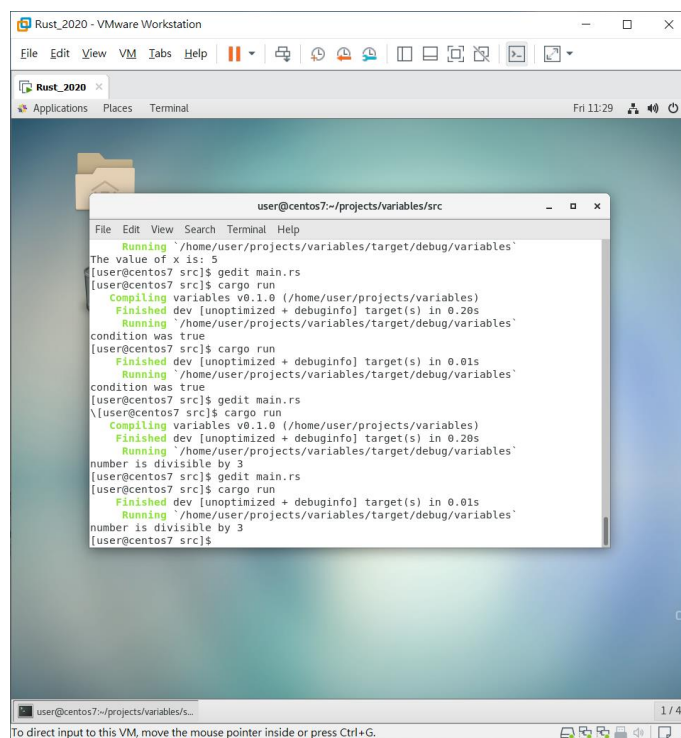
```
File Edit View Search Terminal Help  
2 | let x = 5;  
   | ^ help: if this is intentional, prefix it with an underscore: `x`  
   |  
   | = note: `[warn(unused_variables)]` on by default  
  
warning: 1 warning emitted  
  
Finished dev [unoptimized + debuginfo] target(s) in 0.22s  
Running `/home/user/projects/variables/target/debug/variables`  
The value of y is: 4  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.19s  
Running `/home/user/projects/variables/target/debug/variables`  
The value of x is: 5  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.20s  
Running `/home/user/projects/variables/target/debug/variables`  
condition was true  
[user@centos7 src]$
```

使用 `else` 加入更多條件

程式說明:

```
fn main() {  
    let number = 6; //利用 else 來判斷除 4 之後的餘數  
  
    if number % 4 == 0 {  
        println!("number is divisible by 4");  
    } else if number % 3 == 0 {  
        println!("number is divisible by 3");  
    } else if number % 2 == 0 {  
        println!("number is divisible by 2");  
    } else {  
        println!("number is not divisible by 4, 3, or 2");  
    }  
}
```

執行結果:



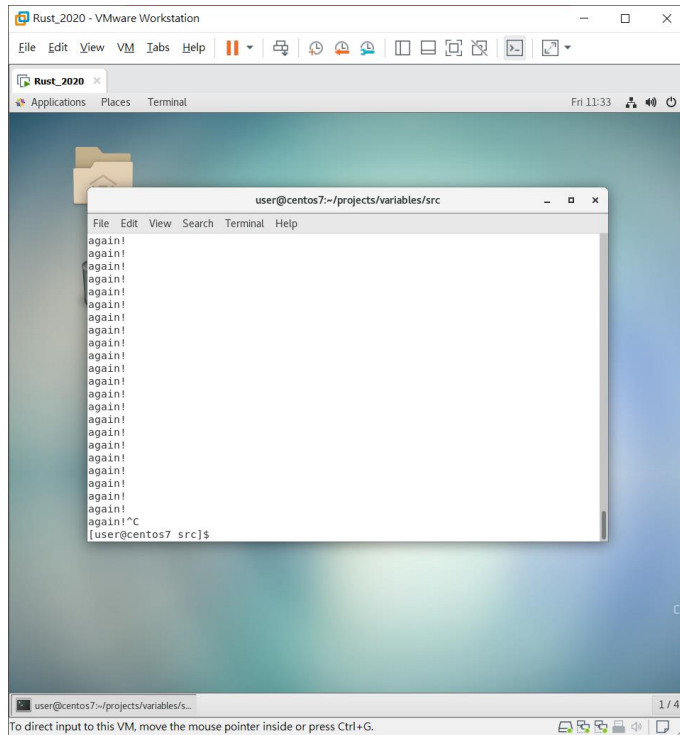
```
user@centos7:~/projects/variables/src  
File Edit View Search Terminal Help  
Running '/home/user/projects/variables/target/debug/variables'  
The value of x is: 5  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.20s  
Running '/home/user/projects/variables/target/debug/variables'  
condition was true  
[user@centos7 src]$ cargo run  
Finished dev [unoptimized + debuginfo] target(s) in 0.01s  
Running '/home/user/projects/variables/target/debug/variables'  
condition was true  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.20s  
Running '/home/user/projects/variables/target/debug/variables'  
number is divisible by 3  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Finished dev [unoptimized + debuginfo] target(s) in 0.01s  
Running '/home/user/projects/variables/target/debug/variables'  
number is divisible by 3  
[user@centos7 src]$
```

- Rust 的三種迴圈：loop、while 和 for

loop

```
fn main() {  
    loop {  
        println!("again!");  
    }  
}
```

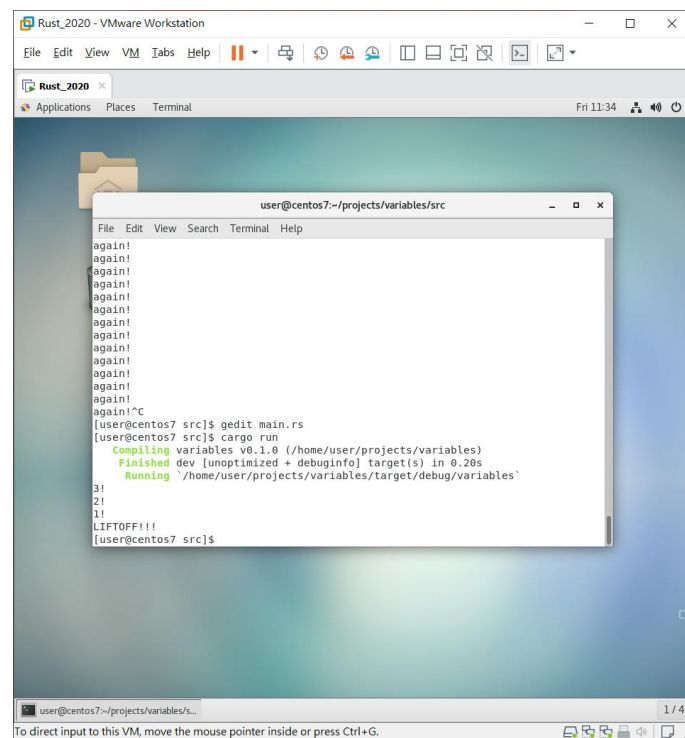
執行結果：





程式說明:

執行結果:

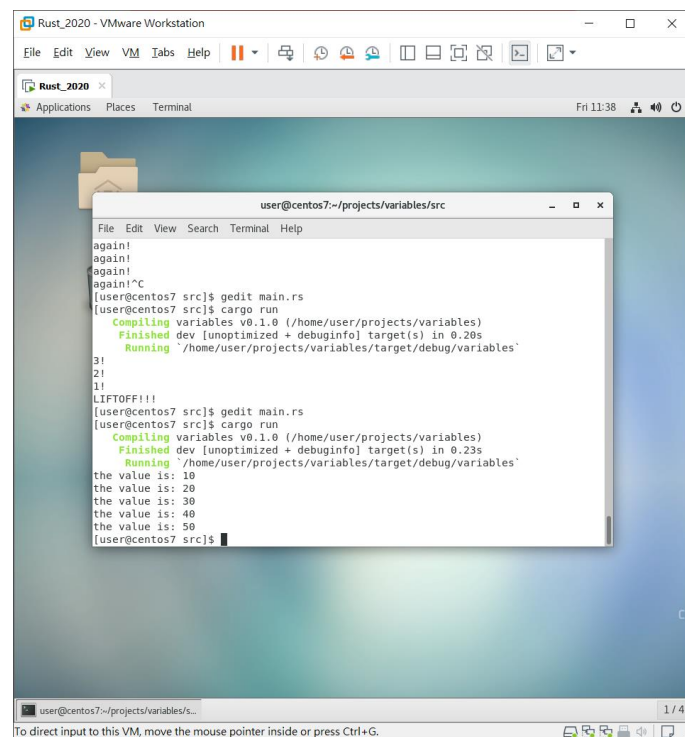


for

程式說明:

```
fn main() {  
    let a = [10, 20, 30, 40, 50];  
  
    for element in a.iter() { //讀取 a.iter 中的每個元素  
        println!("the value is: {}", element);  
    }  
}
```

執行結果:



The screenshot shows a terminal window titled "Rust\_2020 - VMware Workstation" with a sub-window titled "user@centos7:~/projects/variables/src". The terminal displays the following commands and output:

```
again!  
again!  
again!  
again!  
again!  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.20s  
Running /home/user/projects/variables/target/debug/variables  
31  
21  
11  
LIFTOFF!!!  
[user@centos7 src]$ gedit main.rs  
[user@centos7 src]$ cargo run  
Compiling variables v0.1.0 (/home/user/projects/variables)  
Finished dev [unoptimized + debuginfo] target(s) in 0.23s  
Running /home/user/projects/variables/target/debug/variables  
the value is: 10  
the value is: 20  
the value is: 30  
the value is: 40  
the value is: 50  
[user@centos7 src]$
```

參考資料

<https://kaisery.gitbooks.io/trpl-zh-cn/content/>