□□□□ प्रोग्रामिंग का प्रयास करें

□□□□ पिछले कुछ सालों में काफी लोकप्रिय हो चुका एक प्रोग्रामिंग भाषा है। 2006 में, □□□□□□□□ के एक कर्मचारी ने एक व्यक्तिगत प्रोजेक्ट शुरू किया, जिसे बाद में कंपनी का समर्थन मिला और 2010 में इस प्रोजेक्ट को जारी किया गया। इस प्रोजेक्ट का नाम □□□□ रखा गया।

अगला कदम, □□□□ का पहला प्रोग्राम चलाना है। आधिकारिक वेबसाइट खोलें और देखें कि प्रोग्राम को कैसे चलाया जाता है।

आधिकारिक वेबसाइट ने एक स्क्रिप्ट प्रदान की है:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

(यह कोड ब्लॉक है और इसे अनुवादित नहीं किया जाना चाहिए।)

□□□ पर भी □□□ सिस्टम के पैकेज मैनेजमेंट टूल □□□□□□□□□ का उपयोग करके इंस्टॉल किया जा सकता है। निम्नलिखित कमांड चलाएं:

```
brew install rust
```

मैं यहाँ □□□□ को इंस्टॉल करने के लिए □□□□□□□□□ का उपयोग कर रहा हूँ। इंस्टॉल होने के समय, हम आधिकारिक वेबसाइट को देखते रहें।

अगले हम देखते हैं कि आधिकारिक वेबसाइट पर □□□□□ नामक एक चीज़ दिखाई देती है, जो □□□□ का बिल्ड टूल और पैकेज मैनेजमेंट टूल है।

आधिकारिक वेबसाइट पर कहा गया है:

- □ अपने प्रोजेक्ट को बिल्ड करने के लिए `cargo build` का उपयोग करें
- □ अपने प्रोजेक्ट को चलाने के लिए `cargo run` का उपयोग करें
- □ अपने प्रोजेक्ट को टेस्ट करने के लिए `cargo test` का उपयोग करें

हमें बताएं कि □□□□□ प्रोग्राम को कैसे बनाएं, चलाएं और परीक्षण करें।

चलाएं:

```
brew install rust
```

(नोट: यह कमांड □□□□□ पर □□□□ प्रोग्रामिंग लैंग्वेज को इंस्टॉल करने के लिए □□□□□□□□□ का उपयोग करती है। इसे हिंदी में अनुवादित करने की आवश्यकता नहीं है क्योंकि यह एक टर्मिनल कमांड है और इसे वैसे ही रहना चाहिए।)

आउटपुट:

```
==>            https://homebrew.bintray.com/bottles/rust-1.49.0_1.big_sur.bottle.tar.gz
==>            https://d29vzk4ow07wi7.cloudfront.net/5a238d58c3fa775fed4e12ad74109deff54a82a06cb6a3a4f5
######################################################################## 100.0%
==>   -1.49.0_1.big_sur.bottle.tar.gz
==>
               :
  /usr/local/etc/bash_completion.d
==>
   /usr/local/Cellar/rust/1.49.0_1: 15,736    , 606.2MB
```

यह सफलतापूर्वक स्थापित हो गया है।

जब टर्मिनल में `cargo` चलाया जाता है, तो आउटपुट निम्नलिखित होता है:

उपयोग: □□□□□ [विकल्प] [उपकमांड]

□□□□□□□□: -□, –□□□□□□□□ संस्करण जानकारी प्रिंट करें और बाहर निकलें –□□□□ इंस्टॉल किए गए कमांड्स की सूची दिखाएं –□□□□□□□□ `rustc --explain CODE` चलाएं -□, –□□□□□□□□ विस्तृत आउटपुट का उपयोग करें (-□□ बहुत विस्तृत/□□□□□.□□ आउटपुट) -□, –□□□□□ □□□□□□□ पर कोई आउटपुट प्रिंट न करें –□□□□□ रंग: □□□□, □□□□□□□, □□□□□ –□□□□□□□ □□□□□.□□□□ और कैश को अप-टू-डेट होने की आवश्यकता है –□□□□□□□ □□□□□.□□□□ को अप-टू-डेट होने की आवश्यकता है –□□□□□□□□ नेटवर्क तक पहुंच के बिना चलाएं -□ ... □□□□□ के लिए अस्थिर (नाइटली-ओनली) फ्लैग्स, विवरण के लिए '□□□□□ -□ □□□□' देखें -□, –□□□□ मदद जानकारी प्रिंट करें

कुछ सामान्य कार्गो कमांड्स हैं (सभी कमांड्स देखने के लिए –□□□□ का उपयोग करें): □□□□□, □ वर्तमान पैकेज को कंपाइल करें □□□□□, □ वर्तमान पैकेज का विश्लेषण करें और त्रुटियों की रिपोर्ट करें, लेकिन ऑब्जेक्ट फाइलें न बनाएं □□□□□ टार्गेट डायरेक्टरी को हटाएं □□□ इस पैकेज और इसके डिपेंडेंसीज़ की डॉक्यूमेंटेशन बनाएं □□□ एक नया कार्गो पैकेज बनाएं □□□□ मौजूदा डायरेक्टरी में एक नया कार्गो पैकेज बनाएं □□□, □ लोकल पैकेज का बाइनरी या उदाहरण चलाएं □□□□, □ टेस्ट चलाएं □□□□□ बेंचमार्क चलाएं □□□□□□ □□□□□.□□□□ में सूचीबद्ध डिपेंडेंसीज़ को अपडेट करें □□□□□□ रजिस्ट्री में क्रेट्स खोजें □□□□□□□ इस पैकेज को पैकेज करें और रजिस्ट्री में अपलोड करें □□□□□□□ एक □□□□ बाइनरी इंस्टॉल करें। डिफ़ॉल्ट लोकेशन □□□□□/.□□□□□/□□□ है □□□□□□□□□ एक □□□□ बाइनरी को अनइंस्टॉल करें

किसी विशिष्ट कमांड के बारे में अधिक जानकारी के लिए '□□□□□ □□□□ ' देखें।

`build` `run`

:

```c
```

Rust , Cargo

```
cargo new hello-rust
```

यह `hello-rust` नामक एक नया डायरेक्टरी जनरेट करेगा, जिसमें निम्नलिखित फाइलें होंगी:

□□□□□-□□□□ |- □□□□□.□□□□ |- □□□ |- □□□□.□□ □□□□□.□□□□ रस्ट के लिए मेनिफेस्ट फ़ाइल है। यह वह जगह है जहां आप अपने प्रोजेक्ट के लिए मेटाडेटा और डिपेंडेंसीज़ रखते हैं।

`src/main.rs` वह जगह है जहाँ हम अपना एप्लिकेशन कोड लिखेंगे।

```
$ cargo new hello-rust
```

बाइनरी (एप्लिकेशन) `hello-rust` पैकेज बनाया गया

VSCode

main.rs: (Rust ,)

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

(यह कोड □□□□ प्रोग्रामिंग भाषा में लिखा गया है और इसमें "□□□□□, □□□□□!" प्रिंट करने का काम किया जाता है। इसे हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह एक प्रोग्रामिंग कोड है।)

अगला, स्वाभाविक रूप से प्रोग्राम को बिल्ड और रन करने के बारे में सोचना चाहिए।

```
$ cargo build
```

त्रुटि: `/Users/lzw/ideas/curious-courses/program/run/rust` या किसी भी पैरेंट डायरेक्टरी में `Cargo.toml` नहीं मिला

? cargo , - `cd hello-rust`

, ,

```shell
$ cargo run

   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
    Finished dev [unoptimized + debuginfo] target(s) in 4.43s
     Running `target/debug/hello-rust`
Hello, world!
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया जाता है क्योंकि इसमें कोड और कमांड शामिल हैं जो भाषा-अज्ञेय हैं।)

ठीक है, सफल हो गया। स्ट्रिंग आउटपुट हुआ, और प्रोग्राम काम करने लगा है।

प्रोग्राम में बदलाव करने का प्रयास करें।

```rust
fn main() {
    println!(2+3);
}
```

यह कोड □□□□ प्रोग्रामिंग भाषा में लिखा गया है। इसमें `main` फ़ंक्शन है जो प्रोग्राम का एंट्री पॉइंट है। `println!` मैक्रो का उपयोग करके 2+3 का योग प्रिंट किया जा रहा है। हालांकि, यह कोड सही नहीं है क्योंकि `println!` मैक्रो को स्ट्रिंग फॉर्मेट की आवश्यकता होती है। सही कोड इस प्रकार होगा:

```rust
fn main() {
    println!("{}", 2 + 3);
}
```

इस कोड में, `{}` एक प्लेसहोल्डर है जो `2 + 3` के मान को प्रिंट करेगा।

`cargo run` के बाद, यह दिखाई दिया:

```
   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
  :
 --> src/main.rs:2:14
  |
2 |     println!(2+3);
  |              ^^^
  |
   :
  |
```

```
2 |     println!("{}", 2+3);
  |              ^^^^^
```

त्रुटि: पिछली त्रुटि के कारण प्रक्रिया रद्द की जा रही है

त्रुटि: `hello-rust` को संकलित नहीं किया जा सका

अधिक जानने के लिए, कमांड को `--verbose` के साथ फिर से चलाएं।

Rust ,

```rust
fn main() {
    println!("{}", 2+3);
}
```

यह कोड □□□□ प्रोग्रामिंग भाषा में लिखा गया है। यह एक साधारण प्रोग्राम है जो `2 + 3` का योग करता है और परिणाम को प्रिंट करता है। `println!` मैक्रो का उपयोग करके आउटपुट को कंसोल पर प्रदर्शित किया जाता है।

इस बार सही कर दिया, और वास्तव में 5 आउटपुट हुआ।

बिल्ड (□□□□□) क्या होगा?

```
$ cargo build
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
```

क्यों `build` की आवश्यकता होती है? क्योंकि हो सकता है कि हम केवल एक □□□□□□□□□□ प्रोग्राम जनरेट करना चाहते हों, लेकिन उसे चलाना नहीं चाहते। कुछ बड़े प्रोग्राम के लिए, उन्हें चलाना समय लेने वाला हो सकता है। हो सकता है कि हम उसे लोकल पर जनरेट करना चाहते हों और फिर उसे रिमोट सर्वर पर ट्रांसफर करके वहां चलाना चाहते हों।

हमने □□□□ प्रोग्राम को चला दिया है। अब हमें □□□□ भाषा के और अधिक सिंटैक्स से परिचित होना है, ताकि हम "कंप्यूटर साइंस के रहस्यों को सुलझाने" में बताए गए चर (□□□□□□□□□□), फ़ंक्शन (□□□□□□□□□□), फ़ंक्शन कॉल (□□□□□□□□□ □□□□□□), और एक्सप्रेशन (□□□□□□□□□□□□□) जैसी अवधारणाओं के लिए □□□□ में संबंधित प्रतीकों को पहचान सकें।

छोटा अभ्यास

- □ ऊपर दिए गए उदाहरण की तरह, छात्र अपने कंप्यूटर पर □□□□ प्रोग्रामिंग का प्रयास कर सकते हैं।
- □ अभ्यास पूरा करने के बाद, एक सौ शब्दों के भीतर सारांश या इस लेख के लिए अतिरिक्त जानकारी जमा कर सकते हैं।
