## שלום עולם!

כעת, משראסט מותקנת, הגיע הזמן לכתוב את תוכנית הראסט הראשונה שלכם! מקובל, כאשר לומדים שפת תכנות חדשה, לכתוב תוכנית קטנה המדפיסה את הטקסט `Hello, world!` על המסך, וכך נעשה מייד.

> הערה: ספר זה מניח הכרות בסיסית עם שורת הפקודה. לראסט אין דרישות מיוחדות בכל הנוגע לתוכנה בה אתם מעדיפים לערוך קבצים, או בנוגע לאיזה כלי פיתוח אתם מעדיפים, או היכן אתם מאחסנים את הקוד שלכם. לכן, אם אתם מעדיפים להשתמש בסביבת פיתוח אינטגרטיבית (IDE) במקום בשורת הפקודה, הרגישו חופשיים לעשות זאת. לסביבות IDE רבות יש כבר תמיכה מובנית בראסט; בדקו את תיעוד ה-IDE שלכם לפרטים נוספים. צוות ראסט התמקד בלאפשר תמיכה מעולה ב-IDE באמצעות `rust-analyzer`. ראו [נספח ד'][devtools]<!-- ignore --> לפרטים נוספים.

### יצירת תיקיית פרוייקט

התחילו ביצירת תיקייה לאחסון קוד הראסט שלכם. עבור ראסט אין זה משנה היכן הקוד שלכם נמצא, אבל עבור התרגילים והפרוייקטים בספר זה, מומלץ ליצור תיקיה בשם \*projects\* בתיקיית הבית שלכם, ולשמור את כל הפרוייקטים שם.

פתחו טרמינל והקלידו את הפקודות הבאות כדי ליצור את התיקייה \*projects\* ותיקייה עבור הפרוייקט "!Hello, world" בתוך תיקיית הפרוייקטים.

בסביבות Linux, macOS, ו-PowerShell על Windows, הקלידו:

```console

$ mkdir ~/projects

$ cd ~/projects

$ mkdir hello\_world

$ cd hello\_world

```

עבור Windows CMD הקלידו:

```cmd

> mkdir "%USERPROFILE%\projects"

> cd /d "%USERPROFILE%\projects"

> mkdir hello\_world

> cd hello\_world

```

### כתיבת והרצת תוכנית ראסט

עכשיו, צרו קובץ מקור חדש בשם \*main.rs\*. קבצי ראסט מסתיימים תמיד בסיומת \*rs.\* ללא יוצא מן הכלל. אם אתם משתמשים ביותר ממילה אחת בשם הקובץ, המוסכמה היא להשתמש במקף תחתי כדי להפריד בין המילים. למשל, מקובל לכתוב \*hello\_world.rs\* ולא \*helloworld.rs\*.

כעת, פתחו את הקובץ \*main.rs\* שיצרתם והקלידו לתוכו את הקוד מרשימה 1-1.

<span class="filename">Filename: main.rs</span>

```rust

fn main() {

println!("Hello, world!");

}

```

<span class="caption">רשימה 1-1: תוכנית המדפיסה `Hello, world!`</span>

שמרו את הקובץ וחזרו בחלון הטרמינל לתיקייה \*~/projects/hello\_world\*. על Linux או macOS, הקלידו את הפקודות הבאות בכדי לקמפל ולהריץ את הקובץ:

```console

$ rustc main.rs

$ ./main

Hello, world!

```

על Windows, הקלידו את הפקודה `./main.exe` במקום `./main`:

```powershell

> rustc main.rs

> .\main.exe

Hello, world!

```

ללא קשר לסוג מערכת ההפעלה שלכם, המחרוזת `Hello, world!` אמורה להופיע בטרמינל. אם אתם לא רואים את הקלט, פנו חזרה לסעיף ["פתירת בעיות"][troubleshooting]<!-- ignore --> מהחלק על התקנת ראסט כדי לקבל עזרה.

אם `Hello, world!` כן נכתב, כל הכבוד! כתבתם את תכנית הראסט הראשונה שלכם. בכך הפכתם למתכנתי ראסט -- ברוכים הבאים!

### אנטומיה של תכנית ראסט

הבה נסקור את תכנית ה-"!Hello, world" ביתר פירוט. הינה פיסת הפזל הראשונה:

```rust

fn main() {

}

```

שורות אלה מגדירות פונקציה בשם `main`. `main` היא פונקציה מיוחדת: זו תמיד פיסת הקוד הראשונה שרצה בכל קובץ הרצה של תוכנת ראסט. כאן, השורה הראשונה מצהירה על פונקציה בשם `main` שאין לה פרמטרים ושאינה מחזירה כלום. במידה והיו פרמטרים, הם היו מופיעים בין סוגריים עגולים - `()`.

גוף הפונקציה מוקף בסוגריים מסולסלים -`{}`. ראסט דורשת סוגריים מסולסלים סביב הגוף של פונקציות. הסגנון שנחשב כנכון הוא למקם את הסוגר המסולסל הפותח באותה שורה בה מוצהרת הפונקציה, עם רווח יחידי באמצע.

> אם אתם מעוניינים לשמור על סגנון סטנדרטי על פני פרוייקטי הראסט שלכם, אז אתם יכולים להשתמש במפרמט אוטומטי שנקרא `rustfmt` כדי לעצב את הקוד שלכם בסגנון מסויים (עוד על `rustfmt` תמצאו ב-[נספח ד'][devtools]<!-- ignore -->). צוות הראסט הוסיף כלי זה עם ההפצה הסטנדטית של ראסט, בדיוק כמו `rustc`, כך שהוא כבר צריך להיות מותקן על המחשב שלכם!

גוף הפונקציה `main` מכילה את הקוד הבא:

```rust

println!("Hello, world!");

```

שורה זו עושה את כל העבודה בתכנית זעירה זו: היא מדפיסה טקסט על המסך. יש כאן ארבעה פרטים חשובים לציון.

ראשית, הסגנון בראסט הוא להזיח (indent) עם ארבעה רווחים, לא עם טאב.

שנית, `println!` קורא למאקרו של ראסט. שימו לב – זהו מאקרו, ולא פונקציה. אם זו הייתה קריאה לפונקציה, הקוד היה נכתב `println` (ללא הסימן `!`). אנו נדון במאקרואים בראסט בפירוט בפרק 19. בינתיים, כל שעליכם לדעת הוא ששימוש ב-`!` משמעו שאתם קוראים למאקרו ולא לפונקציה רגילה, ושמקרואים לא תמיד מצייתים לאותם הכללים לפיהם מתנהגות פונקציות.

שלישית, התבוננו במחרוזת `"Hello, world!"`. מחרוזת זו מועברת כארגומנט ל-`println!`, והמחרוזת מודפסת על המסך.

הנקודה הרביעית היא הנקודה-פסיק (`;`) בסוף השורה, אשר מציינת שביטוי זה הסתיים והבא אחריו מוכן להתחיל. רוב שורות הקוד בראסט מסתיימות בנקודה-פסיק.

### קמפול והרצה הם צעדים נפרדים

זה עתה הרצתם תוכנית חדשה, אז הבה נבחן כל צעד בתהליך.

לפני הרצת תכנית ראסט, יש לקמפל אותה באמצעות קומפיילר ראסט באמצעות הפקודה `rustc` והעברת שם קובץ המקור, כך:

```console

$ rustc main.rs

```

אם יש לכם רקע ב-C או ++C, וודאי תשימו לב לדמיון ל-`gcc` או `clang`. לאחר קומפילציה מוצלחת תפיק ראסט קובץ הרצה בינארי.

על Linux, macOS, ו-PowerShell על Windows, תוכלו לראות את קובץ ההרצה ע"י הקלדת הפקודה `ls` במעטפת שלכם:

```console

$ ls

main main.rs

```

במערכות Linux ו-macOS תראו שני קבצים. על PowerShell על Windows תראו את אותם שלושה קבצים שתראו על CMD. על CMD על Windows תכתבו:

```cmd

> dir /B %= the /B option says to only show the file names =%

main.exe

main.pdb

main.rs

```

זה מראה את קובץ קוד המקור עם התוספת \*rs.\*, קובץ ההרצה (\*main.exe\* על Windows, אבל \*main\* על פלטפורמות אחרות), וגם, כאשר משתמשים ב-Windows, קובץ המכיל מידע לניפוי שגיאות עם תוספת \*.pdb\*. ומרגע זה, מריצים את קובץ ה-\*main\* או main.exe</em>, כך:

```console

$ ./main # or .\main.exe on Windows

```

במידה ו-\*main.rs\* היא תכנית ה-"!Hello, world" שלכם, שורה זו תדפיס `!Hello, world` בטרמינל.

אם אתם מרגישים יותר בנוח עם שפה דינאמית, כמו Ruby, Python, או JavaScript, אתם עשויים שלא להכיר את הקימפול וההרצה כתהליכים נפרדים. ראסט היא דוגמה לשפה מסוג ahead-of-time compiled language. כלומר, זוהי שפה המאפשרת לקמפל תוכנית לקובץ הרצה בינארי כשלב נפרד ושונה מהרצה. כך, לאחר שלב הקימפול אפשר לתת את קובץ ההרצה למשתמש אחר, ובאפשרותם יהיה להריץ את הקובץ אפילו מבלי להתקין ראסט. לעומת זאת, אם תתנו למישהו קובץ מסוג \*rb\*, \*.py.\*, או \*js.\*, אז, כדי להריץ את הקובץ, הם ייאלצו להתקין Ruby, Python, או JavaScript (בהתאמה). מצד שני, בשפות אלה מספיקה פקודה אחת בלבד כדי להריץ את התוכנית. כשזה מגיע לתכנון שפות תכנות, הכל שאלה של תן-וקח.

קימפול בלבד באמצעות `rustc` זה טוב ויפה עבור תוכנות פשוטות, אבל ככל שהפרוייקט שלכם גדל, כך תרצו לנהל את כל האופציות ולהקל לשתף את הקוד שלכם. בשלב הבא נכיר לכם את הכלי קארגו, אשר יסייע לכם בכתיבת תוכנות ראסט אמיתיות.

[troubleshooting]: ch01-01-installation.html#troubleshooting

[devtools]: appendix-04-useful-development-tools.md