

Programming Languages (3)
Going outside Jupyter
and Using Libraries

Kenjiro Taura

Objectives

- ▶ make programs outside Jupyter playground
 - ▶ SSH (command line)
 - ▶ editors, not web browsers
 - ▶ build system
- ▶ use libraries
- ▶ split a program into multiple files (\approx use something defined in another file)

Build system

many languages have “build system” to help you use external libraries

- ▶ Go : `go` is it
- ▶ Julia : no particular build system
- ▶ OCaml : `dune` <https://dune.build/>
- ▶ Rust : `cargo`

Using libraries

using a library entails different procedures depending on how “embedded” it is into the language

- ▶ some libraries are *“builtin”*
 - ▶ automatically available in every program
- ▶ some libraries are *“standard”*
 - ▶ you need to master how to refer to names in it
 - ▶ you say “import” or “use” it and/or use prefixes to refer to names in it
 - ▶ installed with the language
- ▶ some libraries are *“external”*
 - ▶ you may have to install it
 - ▶ you may have to tell the compiler where it is

Importing a library to your program

- ▶ assume M is a module name and n a name defined in M
- ▶ OCaml :
 - ▶ call $M.n$
 - ▶ `open` M and call n
- ▶ Julia :
 - ▶ `import` M and call $M.n$
 - ▶ `using` M and call n
- ▶ Go :
 - ▶ `import` " M " and call $M.n$
- ▶ Rust :
 - ▶ a module may contain a module
 - ▶ assume C is the name of a “crate”
 - ▶ call $C::M_0::M_1:: \dots ::n$
 - ▶ `use` $C::M_0::M_1:: \dots ::n$ and call n
 - ▶ anywhere between the two

Repository of libraries

- ▶ master how to get information you need (names of functions, their types, etc.) from those repositories
- ▶ is it builtin? standard? external?
- ▶ OCaml : opam <https://opam.ocaml.org/>
- ▶ Julia : Julia packages
<https://julialang.org/packages/>
- ▶ Go : <https://pkg.go.dev/>
- ▶ Rust : <https://crates.io/>