DS-210: PROGRAMMING FOR DATA SCIENCE

LECTURE 26

1. MODULES

2. USING PATHS TO ACCESS MODULES

3. IMPORTING PATHS VIA `use`

4. STRUCTS WITHIN MODULES

MODULES

- separate your code into different namespaces
- also put it in separate files

MODULES

- separate your code into different namespaces
- also put it in separate files

WHY MODULES?

MODULES

- separate your code into different namespaces
- also put it in separate files

WHY MODULES?

- limit number of additional identifiers in the main namespace
- organize your codebase into meaningful parts
- hide auxiliary internal code

MODULES

Up to now: **our** functions and structs (mostly) in the same namespace

- **exception:** functions in structs and enums

MODULES

Up to now: **our** functions and structs (mostly) in the same namespace

- **exception:** functions in structs and enums

One can create namespaces, using `mod`

In [2]:
```
mod things_to_say {

    fn say_hi() {
        say("Hi");
    }

    fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

MODULES

Up to now: **our** functions and structs (mostly) in the same namespace

- **exception:** functions in structs and enums

One can create namespaces, using `mod`

In [2]:
```
mod things_to_say {

    fn say_hi() {
        say("Hi");
    }

    fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

You have to use the module name to access a function.

In [ ]:
```
things_to_say::say_hi();
```

MODULES

Up to now: **our** functions and structs (mostly) in the same namespace

- **exception:** functions in structs and enums

One can create namespaces, using `mod`

In [2]:
```
mod things_to_say {

    fn say_hi() {
        say("Hi");
    }

    fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

You have to use the module name to access a function.

In [3]:
```
things_to_say::say_hi();
```

```
things_to_say::say_hi();
                  ^^^^^^ private function
function `say_hi` is private
```

- By default, all definitions in the namespace are private.
- Advantage: one can hide all internally used code
- Use `pub` to make functions or types public

In [4]:
```
mod things_to_say {

    pub fn say_hi() {
        say("Hi");
    }

    pub fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

In [ ]:
```
things_to_say::say_hi();
```

In [ ]:
```
things_to_say::say("abc");
```

- By default, all definitions in the namespace are private.
- Advantage: one can hide all internally used code
- Use `pub` to make functions or types public

In [4]:
```
mod things_to_say {

    pub fn say_hi() {
        say("Hi");
    }

    pub fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

In [5]:
```
things_to_say::say_hi();
```

```
Hi!
```

In [ ]:
```
things_to_say::say("abc");
```

- By default, all definitions in the namespace are private.
- Advantage: one can hide all internally used code
- Use `pub` to make functions or types public

In [4]:
```
mod things_to_say {

    pub fn say_hi() {
        say("Hi");
    }

    pub fn say_bye() {
        say("Bye");
    }

    fn say(what: &str) {
        println!("{}!",what);
    }

}
```

In [5]:
```
things_to_say::say_hi();
```

```
Hi!
```

In [6]:
```
things_to_say::say("abc");
```

```
things_to_say::say("abc");
                  ^^^ private function
function `say` is private
```

NESTING POSSIBLE

In [7]:

```
mod level_1 {

    mod level_2_1 {

        mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

NESTING POSSIBLE

In [7]:
```
mod level_1 {

    mod level_2_1 {

        mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

Specifying the full path to access a function

In [ ]:
```
level_1::level_2_1::level_3::where_am_i();
```

NESTING POSSIBLE

In [7]:
```
mod level_1 {

    mod level_2_1 {

        mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

Specifying the full path to access a function

In [8]:
```
level_1::level_2_1::level_3::where_am_i();
```

```
level_1::level_2_1::level_3::where_am_i();
         ^^^^^^^^^^ private module
module `level_2_1` is private
```

NESTING POSSIBLE

In [7]:
```
mod level_1 {

    mod level_2_1 {

        mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

Specifying the full path to access a function

In [8]:
```
level_1::level_2_1::level_3::where_am_i();
```

```
level_1::level_2_1::level_3::where_am_i();
         ^^^^^^^^^^ private module
module `level_2_1` is private
```

Sub-modules have to be made public too!

NESTING POSSIBLE

Now sub-modules are public.

In [9]:
```
mod level_1 {

    pub mod level_2_1 {

        pub mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    pub mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

In [ ]:
```
level_1::level_2_1::level_3::where_am_i();
```

NESTING POSSIBLE

Now sub-modules are public.

In [9]:
```
mod level_1 {

    pub mod level_2_1 {

        pub mod level_3 {

            pub fn where_am_i() {println!("3");}

        }

        pub fn where_am_i() {println!("2_1");}

    }

    pub mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}
```

In [10]:
```
level_1::level_2_1::level_3::where_am_i();
```

```
3
```

PATHS TO MODULES

```
level_1
|
+--level_2_1
|  |
|  +--level_3
|  |  |
|  |  +--where_am_i
|  |  |
|  |  +--call_someone_else     <== new function
|  |
|  +--where_am_i
|
+--level_2_2
|  |
|  +--where_am_i
|
+--where_am_i

where_am_i     <== new function
```

PATHS TO MODULES

In [11]:
```
mod level_1 {

    pub mod level_2_1 {

        pub mod level_3 {

            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                where_am_i();
            }
        }

        pub fn where_am_i() {println!("2_1");}
    }

    pub mod level_2_2 {

        pub fn where_am_i() {println!("2_2");}

    }

    pub fn where_am_i() {println!("1");}

}

fn where_am_i() {println!("main namespace");}
```

In [12]:
```
level_1::level_2_1::level_3::call_someone_else();
```

```
3
```

PATHS TO MODULES

Global paths: start from `crate`

In [13]:
```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                crate::where_am_i();
                crate::level_1::level_2_2::
                    where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

In [14]:
```
level_1::level_2_1::level_3::call_someone_else();
```

```
main namespace
2_2
```

PATHS TO MODULES

Local paths:

- going one or many levels up via `super`

In [15]:
```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                super::where_am_i();
                super::super::where_am_i();
                super::super::
                    level_2_2::where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

In [16]:
```
level_1::level_2_1::level_3::call_someone_else();
```

```
2_1
1
2_2
```

`use` TO IMPORT THINGS INTO THE CURRENT SCOPE

In [17]:

```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                super::where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

`use` TO IMPORT THINGS INTO THE CURRENT SCOPE

In [17]:
```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                super::where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

Bring a submodule to current scope:

In [18]:
```
use level_1::level_2_1::level_3;
level_3::where_am_i();
```

```
3
```

`use` TO IMPORT THINGS INTO THE CURRENT SCOPE

In [17]:
```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                super::where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

Bring a submodule to current scope:

In [18]:
```
use level_1::level_2_1::level_3;
level_3::where_am_i();
```

```
3
```

Bring a specific function/type to current scope:

In [19]:
```
use level_3::call_someone_else();
call_someone_else();
```

```
2_1
```

`use` TO IMPORT THINGS INTO THE CURRENT SCOPE

In [17]:
```
mod level_1 {
    pub mod level_2_1 {
        pub mod level_3 {
            pub fn where_am_i() {println!("3");}

            pub fn call_someone_else() {
                super::where_am_i();
            }
        }
        pub fn where_am_i() {println!("2_1");}
    }
    pub mod level_2_2 {
        pub fn where_am_i() {println!("2_2");}
    }

    pub fn where_am_i() {println!("1");}
}

fn where_am_i() {println!("main namespace");}
```

Bring a submodule to current scope:

In [18]:
```
use level_1::level_2_1::level_3;
level_3::where_am_i();
```

```
3
```

Bring a specific function/type to current scope:

In [19]:
```
use level_3::call_someone_else();
call_someone_else();
```

```
2_1
```

Bring multiple items to current scope:

In [20]:
```
use level_3::{where_am_i,call_someone_else};
where_am_i();
```

```
3
```

STRUCTS WITHIN MODULES

In [21]:

```
mod test {
    #[derive(Debug)]
    pub struct Point {
       x: i32,
       y: i32,
    }

    impl Point {
        pub fn create(x:i32,y:i32) -> Point {
            Point{x,y}
        }

    }

}
```

STRUCTS WITHIN MODULES

In [21]:
```
mod test {
    #[derive(Debug)]
    pub struct Point {
       x: i32,
       y: i32,
    }

    impl Point {
        pub fn create(x:i32,y:i32) -> Point {
            Point{x,y}
        }

    }

}
```

Accessing a field

In [22]:
```
use test::Point;
let mut p = Point::create(2,3);
println!("{:?}",p);
p.x = 3;
println!("{:?}",p);
```

```
p.x = 3;
  ^ private field
field `x` of struct `Point` is private
```

STRUCTS WITHIN MODULES

Make fields and functions public to be accessible

In [23]:
```
mod test {
    #[derive(Debug)]
    pub struct Point {
       pub x: i32,
       y: i32,
    }

    impl Point {
        pub fn create(x:i32,y:i32) -> Point {
            Point{x,y}
        }

        pub fn update_y(&mut self, y:i32) {
            self.y = y;
        }
    }

}
```

STRUCTS WITHIN MODULES

Make fields and functions public to be accessible

In [23]:
```
mod test {
    #[derive(Debug)]
    pub struct Point {
       pub x: i32,
       y: i32,
    }

    impl Point {
        pub fn create(x:i32,y:i32) -> Point {
            Point{x,y}
        }

        pub fn update_y(&mut self, y:i32) {
            self.y = y;
        }
    }

}
```

Accessing a field

In [24]:
```
use test::Point;
let mut p = Point::create(2,3);
println!("{:?}",p);
p.x = 3;
println!("{:?}",p);
```

```
Point { x: 2, y: 3 }
Point { x: 3, y: 3 }
```

In [25]:
```
p.update_y(2022);
p
```

Out[25]:
```
Point { x: 3, y: 2022 }
```