**Project**

In the course materials, locate and open the saladworks project in VSCode. Alternatively, you can create the project by executing cargo new --lib saladworks.

The instructions are located inside src/lib.rs; they're also written below. Follow the instructions to complete the coding challenge.

The instructions are written as a Rust comment, so you can copy and paste them into a Rust (.rs) file if you want to.

Afterwards, see the following lesson for a solution. The solution may be explained in a video, an article, or both.

1. /\*
2. We're running a salad restaurant! You discover some starter code
3. from a previous developer working at the company. The code includes:
4. - A Vegetable enum
5. - A Protein enum
6. - A Dressing enum
8. Our next goal is to build a Salad struct. Each Salad will have a
9. 'protein', 'vegetables', and a 'dressing' field. A Salad can store
10. 1 protein, any number of vegetables, and 1 dressing. Use a vector
11. to store the vegetables. Derive the Debug trait.
13. We need to implement the following 4 functions/methods on a Salad.
14. All 4 must have a complementary unit test. It's up to you whether you
15. want to write your tests first (TDD) or write your implementation
16. first. Follow the best practices for unit tests (modules, configuration,
17. etc). Feel free to utilize any helper crates (pretty\_assertions,
18. rstest, etc).
20. First, define a 'new' constructor function that accepts a 'protein',
21. a 'vegetables' vector, and a 'dressing' and returns an instance of
22. the Salad. In the test, assert that the 3 fields of the Salad are
23. correctly populated.
25. Next, define an 'is\_valid' method that returns a Boolean. Return
26. a true if a salad has more than 0 vegetables.
28. Next, define a 'calories' method that calculates the total calories
29. in the salad. The Vegetable, Protein, and Dressing enums all support
30. a 'calories' method that return the calories of the item. Remember
31. that 'vegetables' is a vector of multiple Vegetable values -- you'll
32. have to include all of them in your calculation.
34. Finally, define a 'has\_duplicate\_vegetables' method. It should
35. determine if the salad includes any vegetable more than once.
36. Return a Boolean.
37. \*/
39. trait Caloric {
40. fn calories(&self) -> u32;
41. }
43. #[derive(Debug, Clone, Hash, PartialEq, Eq)]
44. enum Vegetable {
45. Tomato,
46. Cucumber,
47. SweetPotato,
48. }
50. impl Caloric for Vegetable {
51. fn calories(&self) -> u32 {
52. match self {
53. Self::Tomato => 20,
54. Self::Cucumber => 15,
55. Self::SweetPotato => 100,
56. }
57. }
58. }
60. #[derive(Debug, Clone, PartialEq, Eq)]
61. enum Protein {
62. CrispyChicken,
63. FriedChicken,
64. Steak,
65. Tofu,
66. }
68. impl Caloric for Protein {
69. fn calories(&self) -> u32 {
70. match self {
71. Self::CrispyChicken => 400,
72. Self::FriedChicken => 500,
73. Self::Steak => 300,
74. Self::Tofu => 200,
75. }
76. }
77. }
79. #[derive(Debug, Clone, PartialEq, Eq)]
80. enum Dressing {
81. Ranch,
82. Vinaigrette,
83. Italian,
84. }
86. impl Caloric for Dressing {
87. fn calories(&self) -> u32 {
88. match self {
89. Self::Ranch => 150,
90. Self::Vinaigrette => 120,
91. Self::Italian => 130,
92. }
93. }
94. }

**Resources for this lecture**

* saladworks.zip