Cairo Crash Course

Advanced data types - Exercise 5

In this exercise we will experiment with an array with multiple types using Enums 😃

Contract Implementation

Implementing a Custom Enum

In `advanced_types_exercise_5.cairo`, follow these instructions:

- Create an Enum named `Data` containing three variants:
  - `Integer` => `u128`
  - `Felt` => `felt252`
  - `Tuple` => `(u32, u32)`
- Implement the `ProcessingImpl` impl for the `Data` Enum, which contains the function `process()`. The function should handle how to print all the `Data` variants:
  - For `Data::Integer(value)`, print "My Super Integer!" followed by `value`
  - For `Data::Felt(value)`, print "Wonderful Felt!" followed by `value`
  - For `Data::Tuple((x, y))`, print "The Amazing Tuple!" followed by `x, y`
- If you are not sure how to implement this trait and its function `process()`, check this.

Creating an array of different types

Complete the skeleton of the function `ultimate_array(get_index: u32, my_array: @Array<Data>) -> Data`. The function takes two arguments: `get_index: u32` representing an array index and a snapshot of `my_array: @Array<Data>`. Inside the function:

- Declare an `if` statement:
  - If the requested index is out of the array bounds, invoke the `panic!` macro with a custom error message of your choice.
- Get the array value at the given index and invoke the `process()` function.
- Return the value from the function.

Writing Tests

In `test_advanced_types_5.cairo`, inside the function `test_ultimate_array()`, follow the next steps to complete the tests:

Invoking the custom type array

- Declare a mutable array explicitly defining its Enum `Data` type.
- Append to the array a `Data::Integer(1337)`.
- Append to the array a `Data::Felt('Cairo rulez!')`.
- Append to the array a `Data::Tuple(('foo', 'bar'))`.

- Call the function `ultimate_array()` with index values 0, 1, and 2, and the array you just created as parameters.
    - Don't forget you need to pass the array as a snapshot!
- Compare your results with the expected output.

Using panic!

- Create a new test function named `test_panic()`.
- Declare a mutable array explicitly defining its Enum `Data` type.
- Append the values of your choice according to the available variants to the array.
- Invoke `ultimate_array()` with an index outside the range of your array elements. For example, if you have three elements in your array, call `ultimate_array(3, @my_array)`.
- Make sure the function panics.

Useful links

Arrays and Enums