[[questions]]
type = "MultipleChoice"
prompt.prompt = """
What can you use to implement a mutable data structure?
"""
prompt.distractors = [
  "`Array<T>`",
  "`Box<T>`",
  "Nothing in particular, just make sure to declare the structure as mutable.",
]
answer.answer = "`Felt252Dict<T>`"
context = """
While Cairo has an immutable memory model, you can use the `Felt252Dict<T>` type to simulate mutable data structures.
"""
id = "43074990-3724-4818-94dd-eeac0c078780"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
When you have a member of type `Felt252Dict<T>` in your generic data structure, what trait do you **must** manually implement for that structure?
"""
prompt.distractors = [
  "`Drop<T>`",
  "None, you can just derive the required traits",
  "`Copy<T>` to read the value of the dictionary",
]
answer.answer = "`Destruct<T>`"
context = """
The member of type `Felt252Dict<T>` can't be dropped so the struct has to implement the `Destruct<T>` trait.
Because the struct is working with generic types, the `Destruct` trait cannot be derived.
The elements stored in the dictionary must implement the `Copy<T>` trait to be read, but the struct type itself doesn't necessarily require this implementation.
"""
id = "76864942-6f11-4fbf-87a3-46480e26749a"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
Consider the following code snippet:
```
struct NullableStack<T> {
    data: Felt252Dict<Nullable<T>>,
    len: usize,
}
trait StackTrait<S, T> {
```

```
    fn push(ref self: S, value: T);
    fn pop(ref self: S) -> Option<T>;
    fn is_empty(self: @S) -> bool;
    fn new() -> S;
}
impl NullableStackImpl<T, +Drop<T>, +Copy<T>> of StackTrait<NullableStack<T>, T> {
    fn push(ref self: NullableStack<T>, value: T) {
        self.data.insert(self.len.into(), NullableTrait::new(value));
        self.len += 1;
    }
    fn pop(ref self: NullableStack<T>) -> Option<T> {
        if self.is_empty() {
            return Option::None;
        }
        self.len -= 1;
        Option::Some(self.data.get(self.len.into()).deref())
    }
    fn is_empty(self: @NullableStack<T>) -> bool {
        *self.len == 0
    }
    // The implementation goes here
}
```
What is the correct implementation of the function `fn new() -> S` ?"""
prompt.distractors = ["""```
fn new() -> NullableStack<T> {
    let data: Felt252Dict<T> = Default::default();
    NullableStack { data, len: 0 }
}
```""", """```
fn new() -> NullableStack<T> {
    NullableStack {
        data: Felt252Dict::default(),
        len: 0,
    }
}
```""", """```
fn new() -> NullableStack<T> {
    NullableStack {
        data: Default::default(),
    }
}
```"""]
answer.answer = """```
fn new() -> NullableStack<T> {
    let data: Felt252Dict<Nullable<T>> = Default::default();
```

```
    NullableStack { data, len: 0 }
}
```"""
context = """
The `new` function should return a `NullableStack<T>` with an empty dictionary of type `Nullable<T>` and a length of 0.
"""
id = "743ccd28-548f-453c-a4a2-edfecc90ed37"