```
[[questions]]
type = "Tracing"
prompt.program = """
#[derive(Copy, Drop)]
enum Location {
    Point : u32,
    Range : (u32, u32)
}
fn main() {
    let loc: Location = Location::Range((2, 5));
    let n: u32 = match loc {
        Location::Point(p) => p,
        Location::Range(r) => match r {
            (0, _) => 0,
            (_, n) => n,
            _ => 404
        },
        _ => 403
    };
    println!("{}", n);
}
"""
answer.doesCompile = false
context = """
Cairo does not support matching for tuples whose members are not enums yet.
As `u32` is not an enum, the matching of `r` can't be done.
"""
id = "b0621230-c040-4f30-b658-14d31f4fab82"
[[questions]]
type = "Tracing"
prompt.program = """
#[derive(Drop)]
enum Either {
    Left: u32,
    Right: ByteArray
}
fn main() {
    let x = Either::Right("Hello World");
    let simple = match x {
      Either::Left(n) => n,
      Either::Right(s) => s.len()
    };
    let doubled = match x {
      Either::Left(n) => n * 2,
      Either::Right(s) => s.len() * 2
    };
```

```
    println!("doubled: {}", doubled);
}
"""
answer.doesCompile = false
context = """
The first match arm `Either::Right(s)` moves the field `s`, so `x` cannot be used in the
second match.
"""
id = "b0147849-6c36-46a5-b933-51289913a621"
[[questions]]
type = "Tracing"
prompt.program = """
fn decr_twice(n: u32) -> Option<u32> {
    match n {
        0 | 1 => Option::None,
        val => Option::Some(val - 2)
    }
}
"""
answer.doesCompile = false
context = """
There's no catch-all pattern in Cairo that allows you to use the value of the pattern.
You have to use the placeholder `_` instead.
"""
id = "bb07c951-7f3d-4225-ae54-adff59774b76"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
Consider this method implemented for the `Option` type:
```
fn unwrap_or<+Drop<T>>(self: Option<T>, default: T) -> T {
    match self {
        Option::Some(x) => x,
        Option::None => default,
    }
}
```
Which sentence best describes the behavior of this function?
"""
prompt.distractors = [
  "Returns a reference to the object inside `self` if it exists, and `default` otherwise",
  "Returns a new option containing the object inside `self` if it exists, and `default`
otherwise",
  "Inserts `default` into `self` if `self` does not already contain a value",
]
answer.answer = "Returns the object inside `self` if it exists, and `default` otherwise"
```

context = """
This function "unwraps" the option by consuming ownership of it and retrieving the value inside, but if no value exists then it falls back by returning `default`. This is a real function in the core library!
"""
id = "72e6696d-379e-4440-af15-803b7255bc80"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
Consider these two implementations of a function to decrement an unsigned number twice.
```
fn decr_twice_v1(n: u32) -> Option<u32> {
    match n {
        0 | 1 => Option::None,
        _ => Option::Some(n - 2)
    }
}
fn decr_twice_v2(n: u32) -> Option<u32> {
    if n == 0 {
        Option::None
    } else if n == 1 {
        Option::None
    } else {
        Option::Some(n - 2)
    }
}
```
The functions have the same behavior for:
"""
prompt.distractors = ["Some, but not all inputs", "No inputs"]
answer.answer = "All inputs"
context = """
The `match` and `if` perform the same operation here. A `match` is like a specialized `if` that checks for equality of the matched object.
"""
id = "e07e8e36-2c53-4b30-8040-091c3d4f2fd1"