```
[[questions]]
type = "Tracing"
prompt.program = """
fn add_suffix(mut s: ByteArray) -> ByteArray {
  s.append(@" world");
  s
}
fn main() {
  let s = "hello";
  let s2 = add_suffix(s);
  println!("{}", s2);
}
"""
answer.doesCompile = true
answer.stdout = "hello world"
context = """
This program is valid because `s` is not used after moving it into `add_suffix`.
"""
[[questions]]
type = "Tracing"
prompt.program = """
fn main() {
  let mut s: ByteArray = "hello";
  let b = false;
  let s2 = s;
  if b {
    foo(s2);
  }
  println!("{}", s);
}
fn foo(mut a: ByteArray){
    a.append(@" world");
}
"""
answer.doesCompile = false
answer.lineNumber = 8
answer.stdout = "hello"
context = """
Because `s` and `s2` are actually the same _variables_ and `s2` could be moved inside
of the if-statement,
it is illegal to use `s` on line 8.
While the if-statement will never execute in this program because `b` is always `false`,
Ciaro does not in general try to determine whether if-statements will or won't execute.
Cairo just
assumes that it *might* be executed, and therefore `s` *might* be moved when calling
`foo(s2)`.
```

"""
[[questions]]
id = "f7d67c11-60bb-4d92-8cc2-8ce82cf4c974"
type = "MultipleChoice"
prompt.prompt = """
Say we have a function that moves an array, like this:
```
fn move_array(a: Array<u32>) {
  // This space intentionally left blank
}
```
Below are four snippets which are rejected by the Cairo compiler.
Imagine that Cairo instead allowed these snippets to compile and run.
Select each snippet that would cause issues at runtime behavior, or select
"None of these snippets" if none of these snippets would cause issues.
"""
answer.answer = ["""
```
let mut a = array![1,2,3];
let a2 = a;
move_array(a);
a.append(1);
```
""", """
```
let mut a = array![1,2,3];
move_array(a);
let a2 = a;
```
"""]
prompt.distractors = ["""
```
let a = array![1,2,3];
let a2 = a;
println!("{}", a[0]);
move_array(a);
```
""", "None of these snippets"]
context = """
The key idea is that when an array is passed to `move_array`, its memory segment can be modified.
Therefore, any attempt to access the array after calling `move_array` is forbidden behavior, as it could write to occupied memory.
However, doing `let a2 = a` and then `println` is not forbidden behavior, as the underlying value can be referred to by many variables.
"""