```
[[questions]]
type = "Tracing"
prompt.program = """
fn incr(ref n: u32) {
    *n += 1;
}
fn main() {
    let mut n : u32 = 1;
    incr(ref n);
    println!("{}", n);
}
"""
answer.doesCompile = false
context = """
The code does not compile because the *desnap* operator has to be used on
snapshots, not references.
Inside the `incr` function, the `n` parameter is a reference to a `u32` value, so the `*n`
operation is not allowed. The code should be changed to `n += 1;`.
"""
id = "6555e9b4-5989-40ee-a8e8-e1fd70d7aa1f"
[[questions]]
type = "Tracing"
prompt.program = """
#[derive(Drop)]
struct Rectangle {
    height: u64,
    width: u64,
}
fn double_rect(ref rect: Rectangle) {
    rect.height *= 2;
    rect.width *= 2;
}
fn main() {
    let rect = Rectangle { height: 10, width: 20 };
    double_rect(ref rect);
    println!("{}", rect.height);
}
"""
answer.doesCompile = false
context = """
The code does not compile because a `ref` can only be used on mutable variables.
You should write `let mut rect = Rectangle { height: 10, width: 20 };` instead.
"""
id = "7237725b-b4d8-4c05-81be-8d8dd4056cc6"
[[questions]]
type = "Tracing"
```

```
prompt.program = """
fn sum(arr: @Array<u128>) -> u128 {
    let mut span = arr.span();
    let mut sum = 0;
    while let Option::Some(x) = span.pop_front() {
        sum += *x;
    };
    sum
}
fn main() {
    let mut arr1: Array<u128> = array![1, 2, 3];
    let snap = @arr1;
    let mut snap2 = snap;
    arr1.append(4);
    snap2 = @arr1;
    println!("{}", sum(snap));
}
"""
answer.doesCompile = true
answer.stdout = "6"
context = """
First `snap` and `snap2` are snapshots of a memory location that contains the array
`arr1`.
Then, `arr1` is mutated by appending a new element.
Finally, `snap2` is updated to be a snapshot of the new array while `snap` remains a
snapshot of the old array.
The function `sum` receives the latter and returns its length before the mutation.
"""
id = "4512ce8b-c183-4ad1-96e9-bb808456c321"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
Choose the working code snippet that properly defines and uses the function to insert a
value at the end of an array while removing the first element and returning it.
The array has to be modified by the function and should be usable after the function call.
"""
prompt.distractors = ["""
```
```
fn give_and_take(arr: @Array<u128>, n: u128) -> u128 {
    arr.append(n);
    arr.pop_front().unwrap_or(0)
}
fn main() {
    let mut arr1: Array<u128> = array![1,2,3];
    let elem = give_and_take(@arr1, 4);
    println!("{}", elem);
```

```
}
```
"""", """
```
fn give_and_take(arr: @Array<u128>, n: u128) -> u128 {
    *arr.append(n);
    *arr.pop_front().unwrap_or(0)
}
fn main() {
    let mut arr1: Array<u128> = array![1,2,3];
    let elem = give_and_take(@arr1, 4);
    println!("{}", elem);
}
```
"""", """
```
fn give_and_take(ref arr: Array<u128>, n: u128) -> u128 {
    arr.append(n);
    arr.pop_front().unwrap_or(0)
}
fn main() {
    let mut arr1: Array<u128> = array![1,2,3];
    let elem = give_and_take(arr1, 4);
    println!("{}", elem);
}
```
"""", """
```
fn give_and_take(ref arr: Array<u128>, n: u128) -> u128 {
    arr.append(n);
    arr.pop_front().unwrap_or(0)
}
fn main() {
    let arr1: Array<u128> = array![1,2,3];
    let elem = give_and_take(ref arr1, 4);
    println!("{}", elem);
}
```
"""]
answer.answer = """
```
fn give_and_take(ref arr: Array<u128>, n: u128) -> u128 {
    arr.append(n);
    arr.pop_front().unwrap_or(0)
}
fn main() {
```

```
    let mut arr1: Array<u128> = array![1,2,3];
    let elem = give_and_take(ref arr1, 4);
    println!("{}", elem);
}
```
"""
context = """
We want to modify the array and return the ownership to the main function so we must use a mutable reference to the array.
The snapshot can't be used because it is immutable and operating on it does not alter the value of the underlying array.
A mutable reference must be used on mutable variables.
To pass a mutable reference as an argument, we must use the `ref` keyword in the definition of the function parameter **and** during the call of the function.
"""
id = "b691d9bd-4047-4eb3-bbc8-096f889bb5ac"