~~~~toml
[[questions]]
type = "Tracing"
prompt.program = """
use core::dict::Felt252Dict;
fn main() {
    let mut balances: Felt252Dict<u64> = Default::default();
    balances.insert('Alex', 100);
    balances.insert('Maria', 200);
    let john_balance = balances.get('John') + 100;
    println!("{}", john_balance);
}
"""
answer.doesCompile = true
answer.stdout = "100"
context = """
The code compiles and runs without panicking because `get` returns the default value
for `u64` when the key is not found.
The value of `john_balance` is `0 + 100 = 100`.
"""
id = "8b538a1f-0cf9-4c57-8304-c4b07e0134dd"
[[questions]]
type = "Tracing"
prompt.program = """
use core::dict::Felt252Dict;
fn main() {
    let mut dict: Felt252Dict<Span<felt252>> = Default::default();
    let a = array![8, 9, 10];
    dict.insert('my_span', a.span());
    let my_span = d.get('my_span');
    println!("{:?}", *my_span.at(0));
}
"""
answer.doesCompile = false
context = """
We could store a Span<T> in the dictionary because it implements the Copy<T> trait.
However, the `zero_default` method is not implemented for Span<T> which thus must
be wrapped inside a Nullable<T> type.
For this reason the code will not compile.
"""
id = "af5bc548-fb84-487f-958e-0622d242dc6f"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
What will be the output of this code snippet?
```
use core::dict::Felt252Dict;
~~~~

~~~
fn main() {
    let mut balances: Felt252Dict<u64> = Default::default();
    balances.insert('Alex', 200);
    balances.insert('Maria', 200);
    balances.insert('Alex', 100);
    let alex_balance = balances.get('Alex') - 150;
    balances.insert('Alex', alex_balance);
    println!("Alex : {}", alex_balance);
}
```
"""
prompt.distractors = ["Alex : -50", "Alex : 0", "Alex : 100"]
answer.answer = "There will be a runtime panic."
context = """
The value type of this dictionary is `u64`, which is an unsigned integer. So the variable
`alex_balance` is an unsigned integer that can't be negative.
The subtraction operation will cause a runtime panic.
"""
id = "8fe876cf-4373-42ca-ae2c-4d13ae23dbed"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
We want to write a function to append a value to an array stored in a dictionary.
Choose the right line of code to make the function below work as expected.
```
fn append_value(ref dict: Felt252Dict<Nullable<Array<u8>>>, key: felt252,
value_to_append: u8) {
    // insert the right line here
    let mut my_array_unboxed = my_array.deref_or(array![]);
    my_array_unboxed.append(value_to_append);
    dict = entry.finalize(NullableTrait::new(my_array_unboxed));
}
```
"""
prompt.distractors = [
  "`let my_array = dict.entry(key);`",
  "`let mut my_array = dict.entry(key);`",
  "None of these options are correct: Arrays can't be mutated inside Dicts.",
]
answer.answer = "`let (entry, my_array) = dict.entry(key);`"
context = """
The `entry` method returns a tuple with the entry and the value.  We can mutate this
value, and then
finalize the entry with this new value, which restores ownership of the dictionary in the
calling
context.
~~~

~~~
"""
id = "f78d9b38-1d3a-4b00-a014-9c618070738c"
[[questions]]
type = "MultipleChoice"
prompt.prompt = """
Let's consider the following instructions and the associated entry table:
```
balances.insert('Alex', 100);
balances.insert('Maria', 200);
balances.insert('John', 300);
balances.insert('Alex', 50);
balances.insert('Maria', 150);
balances.insert('Alicia', 250);
```
After squashing, how many entries will the table contain?
"""
prompt.distractors = ["6", "3", "0"]
answer.answer = "4"
context = """
Squashing only keeps the last entry for each key. In this case, the table will only contain
the entries for 'John', 'Alex', 'Maria', and 'Alicia'.
"""
id = "d643e8df-2b76-4d2a-bb1f-1a00e53ec8df"
~~~