Unrecoverable Errors with `panic`
In Cairo, unexpected issues may arise during program execution, resulting in runtime errors. While the `panic` function from the core library doesn't provide a resolution for these errors, it does acknowledge their occurrence and terminates the program. There are two primary ways that a panic can be triggered in Cairo: inadvertently, through actions causing the code to panic (e.g., accessing an array beyond its bounds), or deliberately, by invoking the `panic` function.
When a panic occurs, it leads to an abrupt termination of the program. The `panic` function takes an array as an argument, which can be used to provide an error message and performs an unwind process where all variables are dropped and dictionaries squashed to ensure the soundness of the program to safely terminate the execution.
Here is how we can call `panic` from inside a program and return the error code `2`:
<span class="filename">Filename: src/lib.cairo</span>
```cairo
{{#include ../listings/ch09-error-handling/no_listing_01_panic/src/lib.cairo}}
```
Running the program will produce the following output:
```shell
{{#include ../listings/ch09-error-handling/no_listing_01_panic/output.txt}}
```
As you can notice in the output, the call to `println!` macro is never reached, as the program terminates after encountering the `panic` statement.
An alternative and more idiomatic approach to panic in Cairo would be to use the `panic_with_felt252` function. This function serves as an abstraction of the array-defining process and is often preferred due to its clearer and more concise expression of intent. By using `panic_with_felt252`, developers can panic in a one-liner by providing a `felt252` error message as an argument, making the code more readable and maintainable.
Let's consider an example:
```cairo
{{#include ../listings/ch09-error-handling/no_listing_02_with_felt252/src/lib.cairo}}
```
Executing this program will yield the same error message as before. In that case, if there is no need for an array and multiple values to be returned within the error, `panic_with_felt252` is a more succinct alternative.
`panic!` Macro
`panic!` macro can be really helpful. The previous example returning the error code `2` shows how convenient `panic!` macro is. There is no need to create an array and pass it as an argument like with the `panic` function.
```cairo
{{#include ../listings/ch09-error-handling/no_listing_03_panic_macro/src/lib.cairo}}
```
Unlike the `panic_with_felt252` function, using `panic!` allows the input, which is ultimately the panic error, to be a literal longer than 31 bytes. This is because `panic!` takes a string as a parameter. For example, the following line of code will successfully

compile:
```cairo, noplayground
panic!("the error for panic! macro is not limited to 31 characters anymore");
```
`nopanic` Notation
You can use the `nopanic` notation to indicate that a function will never panic. Only `nopanic` functions can be called in a function annotated as `nopanic`.
Here is an example:
```cairo,noplayground
{{#include ../listings/ch09-error-handling/no_listing_04_nopanic/src/lib.cairo}}
```
This function will always return `42` and is guaranteed to never panic. Conversely, the following function is not guaranteed to never panic:
```cairo,noplayground
{{#include ../listings/ch09-error-handling/no_listing_05_nopanic_wrong/src/lib.cairo:wrong-nopanic}}
```
If you try to compile this function that includes code that may panic, you will get the following error:
```shell
{{#include ../listings/ch09-error-handling/no_listing_05_nopanic_wrong/output.txt}}
```
Note that there are two functions that may panic here, `assert` and equality with `==`. We usually don't use `assert` function in practice and use `assert!` macro instead. We will discuss `assert!` macro in more detail in the [Testing Cairo Programs][assert macro] chapter.
[assert macro]: ./ch10-01-how-to-write-tests.md#checking-results-with-the-assert-macro
`panic_with` Attribute
You can use the `panic_with` attribute to mark a function that returns an `Option` or `Result`. This attribute takes two arguments, which are the data that is passed as the panic reason as well as the name for a wrapping function. It will create a wrapper for your annotated function which will panic if the function returns `None` or `Err`, with the given data as the panic error.
Example:
```cairo
{{#include ../listings/ch09-error-handling/no_listing_06_panic_with/src/lib.cairo}}
```
{{#quiz ../quizzes/ch09-01-unrecoverable-errors-with-panic.toml}}