Error Handling

- When a function encounters an error condition, it *throws* an error. That function's **caller** can then *catch* the error and do something about it.
- A function indicates that it can throw an error with the `throws` keyword in its declaration.
- When you call a function that can throw, you prepend the call with `try`.

```
func canThrowAnError() throws {
    // this function may or may not throw an error
}
```

- Swift automatically propagates errors out of their scope until they are handled by a `catch` clause.
- A `do` statement creates a new containing scope, allowing errors to be propagated to one or more `catch` clauses.

```
func makeASandwich() throws {
    // ...
}

do {
    try makeASandwich()
    eatASandwich()
} catch Error.OutOfCleanDishes {
    washDishes()
} catch Error.MissingIngredients(let ingredients) {
    buyGroceries(ingredients)
}
```

- If no error is thrown, the `eatASandwich()` function is called.
- If an error is thrown and it matches the `Error.OutOfCleanDishes` case, then the `washDishes()` function will be called.
- If an error is thrown and it matches the `Error.MissingIngredients` case, then the `buyGroceries(_:)` function is called with the associated `[String]` value captured by the catch pattern.