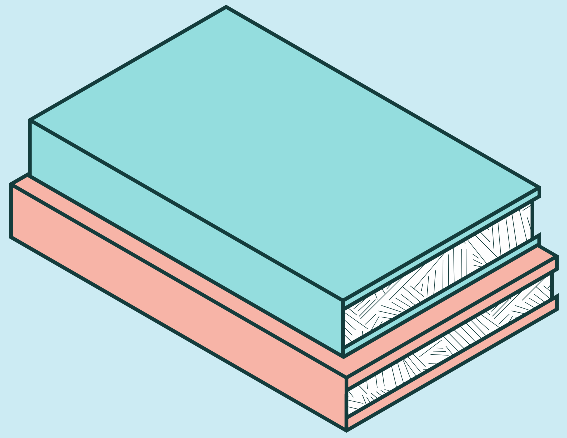
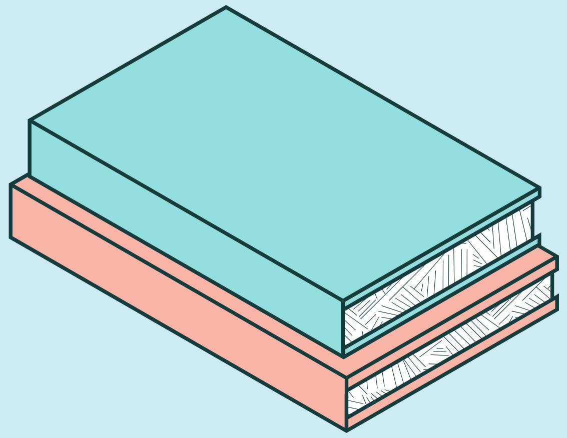
recap



- just like everything else in python, exceptions are objects
- they interrupt control flow when raised and not handled
- SyntaxErrors are pure errors, whereas other exceptions are best thought of as communicating some problematic occurrence that arises during code execution

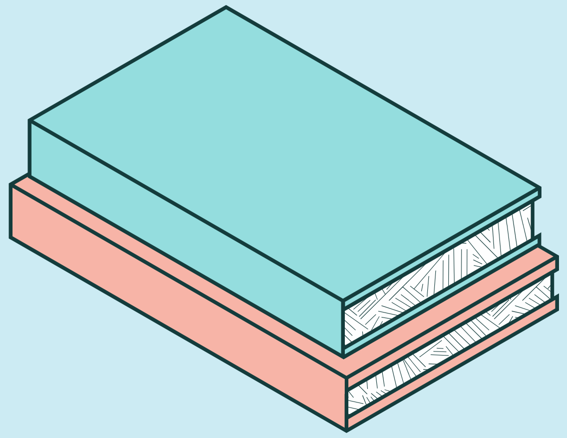
recap



- the exception propagation flow is interrupted by handlers defined in except blocks
- the handlers in turn are always associated with a try block, i.e. they don't stand solo
- exception handlers are specific to each type of exception
- broad catching is a fun way to silence all problems, but not a good idea when writing software

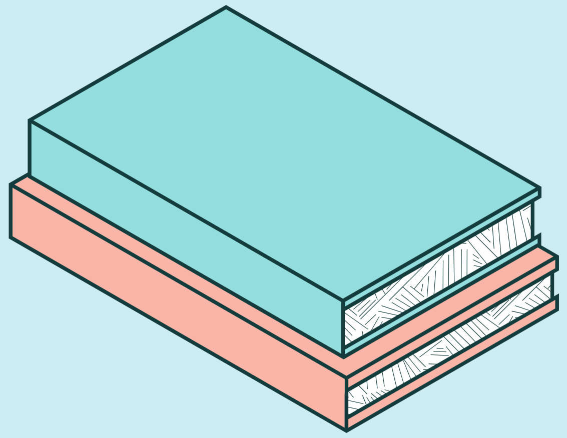
Raising

recap



- exceptions could be explicitly raised using the raise keyword
- this sends the exception up the exception propagation flow, to the next handler if any
- only subclasses of the BaseException class could be raised

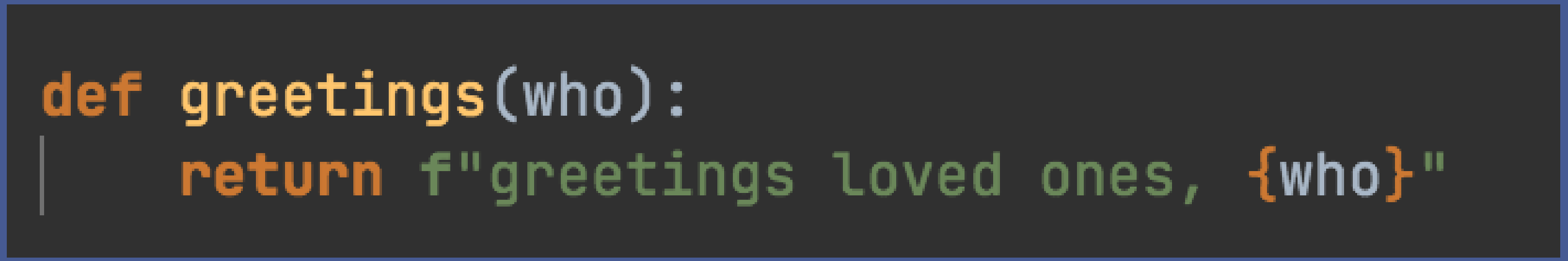
recap



- in python the EAFP (Easier to Ask for Forgiveness than Permission) coding style is preferred and very popular
- the idea is to attempt to carry out an operation and handle the exceptions, if any, afterward
- proficient exception handling is key to enabling EAFP
- this style stands at contrast with LBYL (Look Before You Leap) where the programmer is encouraged to check for the right conditions before attempting an operation

CODE WE WRITE

ahead of time compiled

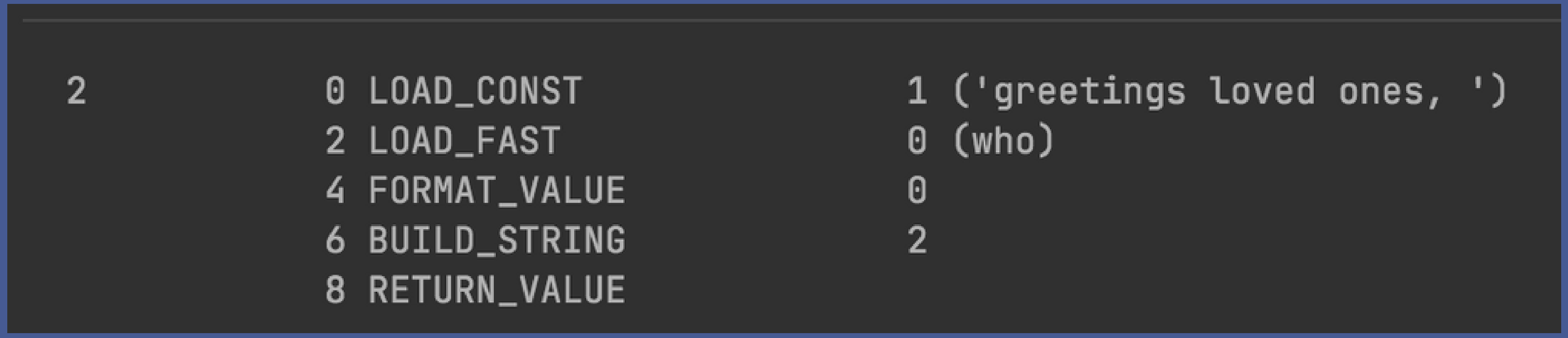


```python
def greetings(who):
    return f"greetings loved ones, {who}"
```

SyntaxError raised here

BYTECODE

to intermediate bytecode



```
2           0 LOAD_CONST               1 ('greetings loved ones, ')
            2 LOAD_FAST                0 (who)
            4 FORMAT_VALUE             0
            6 BUILD_STRING             2
            8 RETURN_VALUE
```

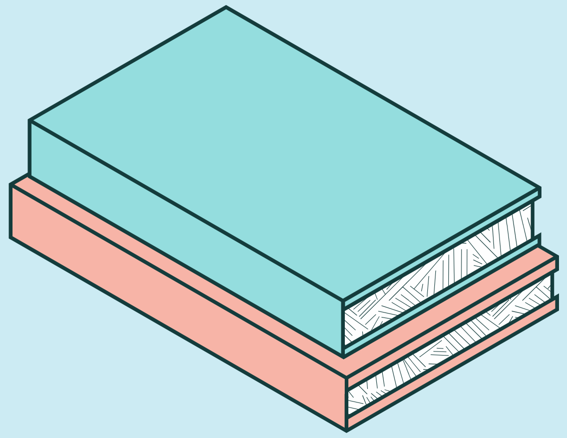
INTERPRETER

finally, interpreted

Other Exceptions raised here

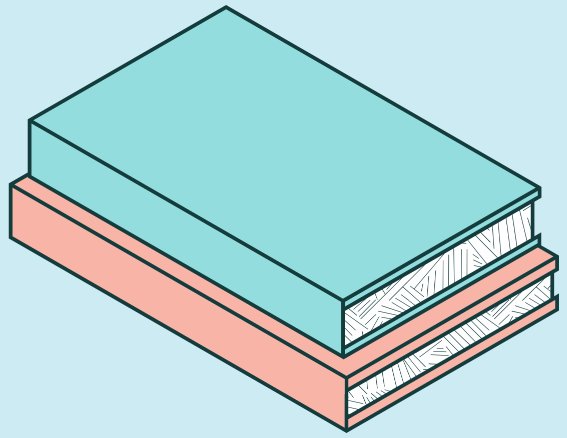
andybek.com

recap



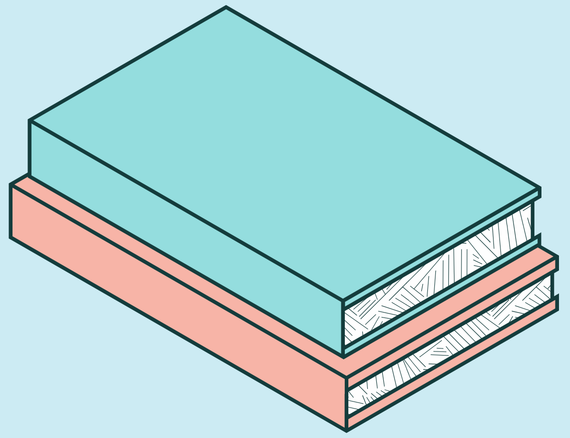
- SyntaxError is a type of exception that cannot normally be caught with regular exception handlers
- the reason is that it interrupts the compilation to bytecode, before any exception handling code is interpreted

recap



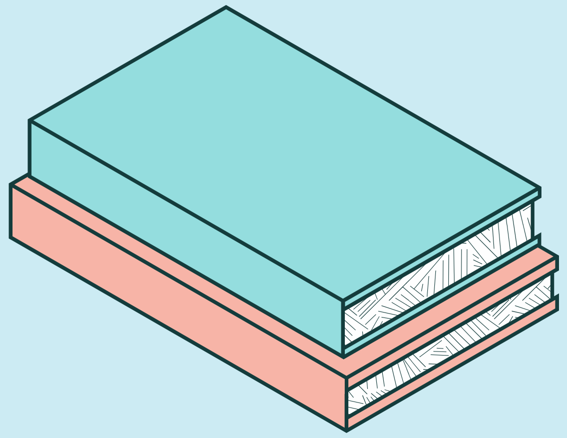
- python defines more than 60 built-in exceptions organized in an inheritance hierarchy with BaseException at the root
- all exceptions in python inherit from BaseException, which has 4 subclasses
- 3 of them (SystemExit, GeneratorExit, KeyboardInterrupt) are process and user-interaction related exceptions that we rarely intend to catch
- all other exceptions inherit from Exception
- when defining multiple exception handlers, we should specify them in increasing order of specificity (i.e. subclasses first)

recap



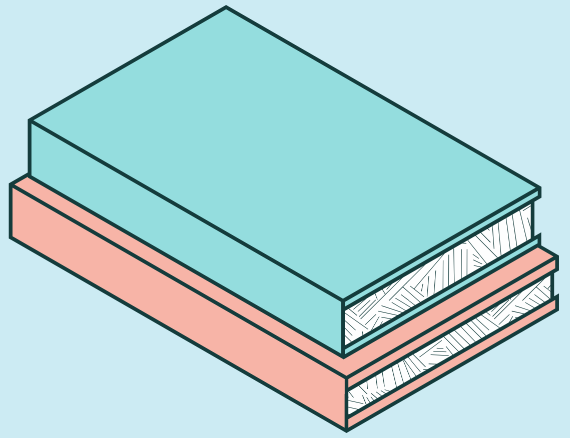
- in addition to try and except, python also supports else blocks
- else only executes when the code in the try does not lead to an exception

recap



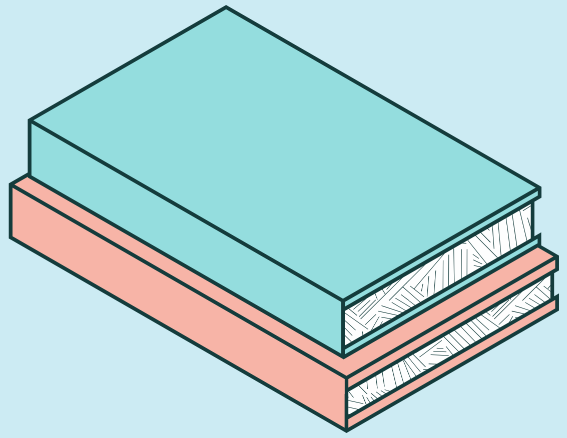
- the finally clause defines blocks of code that execute under all circumstances
- this makes finally ideal for cleanup operations that absolutely need to execute

recap



- exception handlers could be nested within other handlers
- the nested handlers execute within the paused executing context of the outer handlers
- when our handling logic is shared across several exceptions, we could group these exceptions in a single handler

recap



- a class that subclasses BaseException could be raised/handled in the python exception propagation flow
- in practice, it may be a better idea to subclass Exception or one of its descendants when creating custom hierarchies, so as to avoid creating unnecessary siblings of SystemExit, KeyboardInterrupt, and GeneratorExit
- subclassed exceptions allow us to define application-specific hierarchies while also hooking into python's exception propagation flow, via inheritance

Skill Challenge #13

#exceptions

Requirements

> Create a letter guessing game for the English alphabet

> Initially, the computer picks a letter; then, the user is repeatedly given opportunities to guess that letter

> The performance of the user is tracked. Specifically:
- the overall time taken to arrive at an accurate guess, and
- the number of valid guesses that came before what the computer guessed, and
- the number of valid guesses that came after

> Internally, try to have the application use a custom exception hierarchy to refine the handling of the game flow

> In other words, try to have the game control flow incorporate custom exceptions that match the problem domain, e.g. before letter, after letter, not a letter, etc

> In the end end the user gets a summary of how long it took to correctly guess as well how many before/after guesses were made

> If the game is interrupted halfway through (hint: KeyboardInterrupt), the user still gets the summary of the gameplay up to that point, including time played and number of valid guesses of each type