EXCEPTIONS

EXCEPTION

- **exception** = error detected during execution
  - not unconditionally fatal - possible to "handle" them
  - raised by a program either explicitly or by some operation that errors
  - if not handled, result in error messages
  - can also be detected by the program and handled

EXCEPTION EXAMPLES

- Divide by 0 (`ZeroDivisionError`)
- Using an index out of the bounds of a list (`IndexError`)
- Can't find specified file (`FileNotFoundError`)
- Trying to access attribute that doesn't exist (`AttributeError`)
- Trying to perform operation on unsupported type(s) (`TypeError`)

EXCEPTION EXAMPLES (CONT.)

- Trying to access key that doesn't exist in dictionary (`KeyError`)
- Trying to access variable that doesn't exist in this scope (`NameError`)
- Trying to call a function with an invalid value (`ValueError`)

PROCESSING EXCEPTIONS

- do nothing
- handle it where it occurs
- handle it at a different point

UNHANDLED EXCEPTIONS

- program will terminate abnormally
- produces message that describes what occurred and where
- shows "call stack trace" - basically traces back up the path of methods called that caused the exception to occur

TRY-EXCEPT STATEMENTS

```
try:
    //code that might raise exception

except SomeException:
    //do something for this type of exception

except SomeOtherException:
    //do something for this type of exception

else:
    // do something if no exception occurs

finally:
    //code to happen regardless of whether there was exception
```

TRY-EXCEPT STATEMENTS

- at most one `except` clause will be triggered
- always triggers the **first** appropriate where the exception `isinstance` of the specified exception type
- careful with inheritance

TRY-EXCEPT

- `finally` is optional, if exception occurred it is re-raised after finally claus executes
- can have one or more `except`
- upon exception, control transfers to first `except` that corresponds class of exception thrown

EXCEPTION PROPAGATION

- don't necessarily have to handle exception where it occurred
- not handle - control returns to calling method
  - not handled in calling, control returns to method that called method that called...
- exceptions are propagated until handled or passed out of main (uncaught exception)
- handle at higher level by enclosing method invocation with `try-except`