Exceptions

What are exceptions?

- special error objects used to indicate something went wrong
  when normal behavior is impossible or undesired
- exception objects are part of inheritance hierarchy -- all
  are descendants of `BaseException`, most inherit directly from `Exception`
  except `SystemExit` and `KeyboardInterrupt`

Raising Exceptions

```
if not isinstance(value, (int, float):
    raise TypeError('value must be a number')
```

CustomExceptions

- class that inherits from `Exception`
- generally needs no implementation
- can specify message in parentheses when raising
- can get more in depth, but often don't need to

```
class MyException(Exception):
    pass
```

Handling Exceptions

- exceptions indicate abnormal behavior
- some abnormal behavior can be handled in code -- either right where it occurs or elsewhere
- exceptions not handled will cascade up

try-except

```
try:
    # something that might cause an exception
except Exception as e:
    # do something to handle it
    # can have multiple except clauses for different exception types
finally:
    # clean up
    # last task to execute before try completes
    # runs regardless of whether statement produces an exception
```

Different Philosophies

- EAFP: easiert to ask forgiveness than permission
  - assume keys exist, things are right types
  - catches exceptions that occur if assumption is false
  - involves many try-except clauses
- LBYL: look before you leap
  - prechecks before making calls
  - e.g. check if key is in dictionary before trying to access