Python Exceptions

Table of contents

An **exception** is a run-time error — unlike a syntax error, it can't be detected until the offending line actually executes. Left uncaught, it aborts the program.

Common built-in exceptions

Exception	Raised when…
`AssertionError`	an `assert` fails
`IOError`	a file does not exist
`IndexError`	a sequence index is out of range
`KeyError`	a dict key does not exist
`NameError`	a variable/name does not exist
`TypeError`	an unexpected type is given to a function/operator
`ValueError`	correct type, but an inappropriate value
`ZeroDivisionError`	division by zero

Catching exceptions

```
try:
    <code that may raise>
except SomeError:
    <handle SomeError>
except AnotherError:
```

```
    <handle AnotherError>
else:
    <runs only if no exception occurred>
finally:
    <always runs>
```

- **Specific exceptions must be listed before a bare `except:`** — Python enforces this ordering.
- A bare `except:` catches *any* exception, but hides genuine bugs — prefer catching specific exception types.
- Keep `try` blocks short: once an exception is raised inside one, the rest of the block is skipped.

Raising exceptions

```
raise SomeError                  # bare
raise SomeError("message")       # with an explanatory message
```

LBYL vs. EAFP

Two equally valid philosophies for guarding against errors:

- **LBYL** (Look Before You Leap) — check the condition with an `if` before attempting the risky operation.
- **EAFP** (Easier to Ask for Forgiveness than Permission) — attempt the operation inside a `try`, and handle the resulting exception if it fails.