Effective Programming Practices for Economists

Software engineering

How to raise errors?

Janoś Gabler and Hans-Martin von Gaudecker

Raising a built-in exception

```
def convert_lod_to_dol(lod):
    if not isinstance(lod, list):
        msg = f"lod must be a list, not {type(lod)}."
        raise TypeError(msg)
    # ...

>>> convert_lod_to_dol("hello")

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/janos/playground.ipynb Cell 17 line 1
----> 1 convert_lod_to_dol("hello")

/home/janos/playground.ipynb Cell 17 line 4
      2 if not isinstance(lod, list):
      3     msg = f"lod must be a list, not {type(lod)}."
----> 4     raise TypeError(msg)

TypeError: lod must be a list, not <class 'str'>.
```

- Errors are raised with the `raise` keyword
- You can add a custom message to the built-in exceptions

Common built-in errors

- Python has hundreds of built-in exceptions
- You can get very far with two:
  - **TypeError**: An argument to your function has the wrong type
  - **ValueError**: An argument to your function has the correct type but a wrong value
- The full list is in the documentation

fail functions

```python
def convert_lod_to_dol(lod):
    _fail_if_lod_is_not_a_list(lod)
    # ...


def _fail_if_lod_is_not_a_list(lod):
    if not isinstance(lod, list):
        msg = f"lod must be a list, not {type(lod)}."
        raise TypeError(msg)
```

- It is a good idea to put each check into a separate _fail function
- Choose a long and descriptive name
- Define the function at the bottom of your module

Collect errors before raising

```
def _fail_if_list_of_wrong_types(data):
    invalid_rows = []
    for i, row in enumerate(data):
        if not isinstance(row, dict):
            invalid_rows.append(i)

    if invalid_rows:
        report = "The following rows are not dictionaries:\n"
        for i in invalid_rows:
            report += f"  Row {i} has type {type(data[i])}\n"
        raise TypeError(report)
```

- If you have multiple errors, it is annoying to solve them one by one
- If possible, collect multiple errors before raising
- Don’t go too far!

Raising a custom error

```
>>> class NonTabularDataError(Exception):
...     pass

>>> raise NonTabularDataError(
...   "The lists in dol have unequal length"
... )
---------------------------------------------------------
NonTabularDataError  Traceback (most recent call last)
/home/janos/playground.ipynb Cell 18 line 4
      1 class NonTabularDataError(Exception):
      2     pass
----> 4 raise NonTabularDataError(
      5   "The lists in dol have unequal length"
      6 )

NonTabularDataError: The lists in dol have unequal length
```

- Defining a custom error means defining a class that inherits from Exception
- Trade-off
  - Built-in exceptions are familiar
  - Custom exceptions are more explicit