Exception Handling

Handling Errors During the Program Execution

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#python-web

Exceptions vs Errors

Errors vs Exceptions

- **Errors**
  - The whole system will come to a halt
  - The program will not execute
  - Detect them and then make appropriate changes such that they don't happen
- **Exceptions**
  - Raised when internal events disturb the normal flow of the program
  - Detected by run-time executives or by Operating Systems
  - You can deal with them without letting the system come to a halt

Django Exceptions

Disturb the Normal Flow of the Program

How Does a Django Exception Work?

- **Raising an exception** creates an **instance** or a **subclass** of an exception, which describes **the type and the value of the event**
- What can we do?
  - **Raise** the exceptions
  - **Handle** the exceptions
  - Leave them to be handled by the **Django exception handler**

Django Exceptions

- Django raises some of its **own exceptions** as well as standard Python exceptions
- Types of Django exceptions
  - Django **core** exceptions
  - Django **URL resolver** exceptions
  - Django **database** exceptions
  - Etc.

Django Core Exceptions (1)

- `ObjectDoesNotExist`
  - Occurs when the object does not exist
  - Base class for `Model.DoesNotExist` exceptions (raised when `get()` doesn't find any object)
  - Use it to handle `DoesNotExist` exceptions from multiple models

Django Core Exceptions (2)

- `MultipleObjectsReturned` - occurs when a query returns more than one result
  - Occurs when a query returns more than one result
  - Base class for `Model.MultipleObjectsReturned` exceptions (raised when `get()` finds more than one object)
  - Use it to handle the **Model.MultipleObjectsReturned** exceptions from multiple models

Django Core Exceptions (3)

- `ViewDoesNotExist`
  - Raised by `django.urls` when a requested view **does not exist**
- `MiddlewareNotUsed`
  - Occurs when particular middleware is **not used** in the `MIDDLEWARE` section of `settings.py`
- `ValidationError`
  - Occurs when **data validation fails** in forms or model forms

Django URL Resolver Exceptions

- **`Resolver404`**
  - Occurs when **`path()`** does not have a **valid view** to map
  - Raised by the function **`resolve()`**
  - Part of **`Django.http.Http404`** library
  - E.g., when generating a new path in the code, which is not mapped to any view
- **`NoReverseMatch`**
  - Occurs when the user searches a **wrong endpoint**

All Django exceptions: https://docs.djangoproject.com/en/4.1/ref/exceptions/

Handling Exceptions

Why Should We Handle Exceptions?

- It helps to maintain the normal, desired flow of the program even **when unexpected events occur**
- If exceptions are not handled, **programs may crash**, or **requests may fail**
- To **make the user interface robust**, it is important to handle exceptions to prevent the application from unexpectedly crashing and losing data

How Do They Work?

- Exceptions are handled when a clause declares it **received an exception** of the **same type**
  - The exception propagates up until it **reaches the except clause** capable of handling it
  - Or it **goes unhandled**
- The except clause **can stop** the exception from propagating

Example (1)

- Handle the exception and **do something else** with it

The user is not found

```
try:
    user = User.objects.get(username=username)
except User.DoesNotExist:
    user = User.objects.create(username=username)
```

So, we create a new user

Example (2)

- Catch the exception and **re-raise it**

```
try:
    user = User.objects.get(username=username)
except User.DoesNotExist as exc:
    raise User.DoesNotExist(
        "User doesn't exist in the system. "
        "Create a user first!") from exc
```

Raise it from the exception

- Best for adding additional context

Example (3)

- Catch the exception and **report it** using **`logger`**

```
try:
    user = User.objects.get(username=username)
except User.DoesNotExist:
    logger.exception("User doesn't exist and it
remains unset")
```

- Message will be **available on the log record** allowing when processing it to show you a stack trace

Django Exception Handler

- You **do not** have to catch every exception, **only** the ones you **want** and **know how to handle**
  - Django will handle them, like in the case of 404 Error
- Django exception handler work in the **middleware**
- E.g., if the exceptions are thrown **when processing a view**, the Django exception handler will catch them

Invisible Exception Handler

- An **anti-pattern** of dealing with exceptional code (like business code)
  - When you **bury the code** to handle exceptional cases
- It is **simpler** than using exceptions
- However, it makes it **impossible to predict** how your app might fail in the production
  - Does **not document** the exception

```
try:
    do_something()
except:
    pass
```

Django Logger

Elegant and Flexible Debugging

Why Should We Log?

- **Debug** during development
- Know what's happening in production
  - Giving **runtime information**
- **Troubleshooting** made easier
- We could **see each other messages** later on, without going too deeply into the code
- Provide you with **more and better-structured information** about the state and health of your application

Logging

- Django uses and extends **Python's built-in logging module** to perform **system** logging

settings.py

```
LOGGING = {
    ...
}
```

Brief Overview of Python Logging (1)

- Logging provides a set of convenience functions for simple logging usage
  - `debug()`, `info()`, `warning()`, `error()`, `critical()`
- The logging functions are **named after the level** (severity) of the **events** they are used to track: **DEBUG**, **INFO**, **WARNING**, **ERROR**, **CRITICAL** (in increasing order)
- Only events of the **configured (or the default) level and above** will be tracked

Python Logging Example

- If the **logging level** is set to **INFO**, so the **debug()** message will **not appear** on the console

```
import logging
logging.basicConfig(level=logging.INFO)

logging.critical('This message will be shown')
logging.warning('So should this')
logging.debug('This message will not be shown')
logging.error('And this, too')
logging.info('Also, this one')
```

Brief Overview of Python Logging (2)

- **Handlers** send the log records (created by loggers) to the **appropriate destination**
- By default, no destination is set for any logging messages - **you can specify a destination** (a console, a file, or a network socket)
- A logger can have **multiple handlers**, and each handler can have a **different log level** - use to provide different forms of notification depending on the importance of a message

```
logging.FileHandler(filename='example.txt')
```

Brief Overview of Python Logging (3)

- **Filters** provide **additional control** over which log records are passed from logger to handler
  - You can place **additional criteria** on the logging process
  - Filters can be **installed on loggers** or **handlers**
  - Multiple filters can be used in a chain to perform **multiple filtering actions**
- A log record needs to be **rendered as text**, so the **formatters** describe the **exact format** of that text

Configuring Logging in Django (1)

- The configuration is being in **'dictConfig version 1'** format (the only **dictConfig** format version)
- When the **disable_existing_loggers** key is set to **True**, all loggers from the default configuration are **disabled**

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    ...
}
```

Configuring Logging in Django (2)

- Define **one formatter** called simple, that outputs the log **time**, the log **level name** (e.g., DEBUG) and the log **message**

```
LOGGING = {
    ...
    'formatters': {
        'simple': {
            'format': '{asctime} {levelname} {message}',
            'style': '{',
        },
    },
    ...
```

The format string is merged with str.format()

Configuring Logging in Django (3)

- Define **two filters** which uses the **default logging configuration** for logging when **DEBUG** is **True** or **False**

```
LOGGING = {
    ...
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse',
        },
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    ...
```

Configuring Logging in Django (4)

```
LOGGING = {
    ...
    'handlers': {
        'console': {
            'level': 'INFO',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        },
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler',
            'filters': ['require_debug_false ']
        }
```

Print any INFO (or higher) message to sys.stderr

Emails any ERROR (or higher) message to the admin site

Configuring Logging in Django (5)

- Django provides several **built-in loggers** like **`django`** and **`django.request`**

```
LOGGING = {
    ...
    'loggers': {
        'django': {
            'handlers': ['console'],
            'propagate': True,
        },
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': False,
    },
```

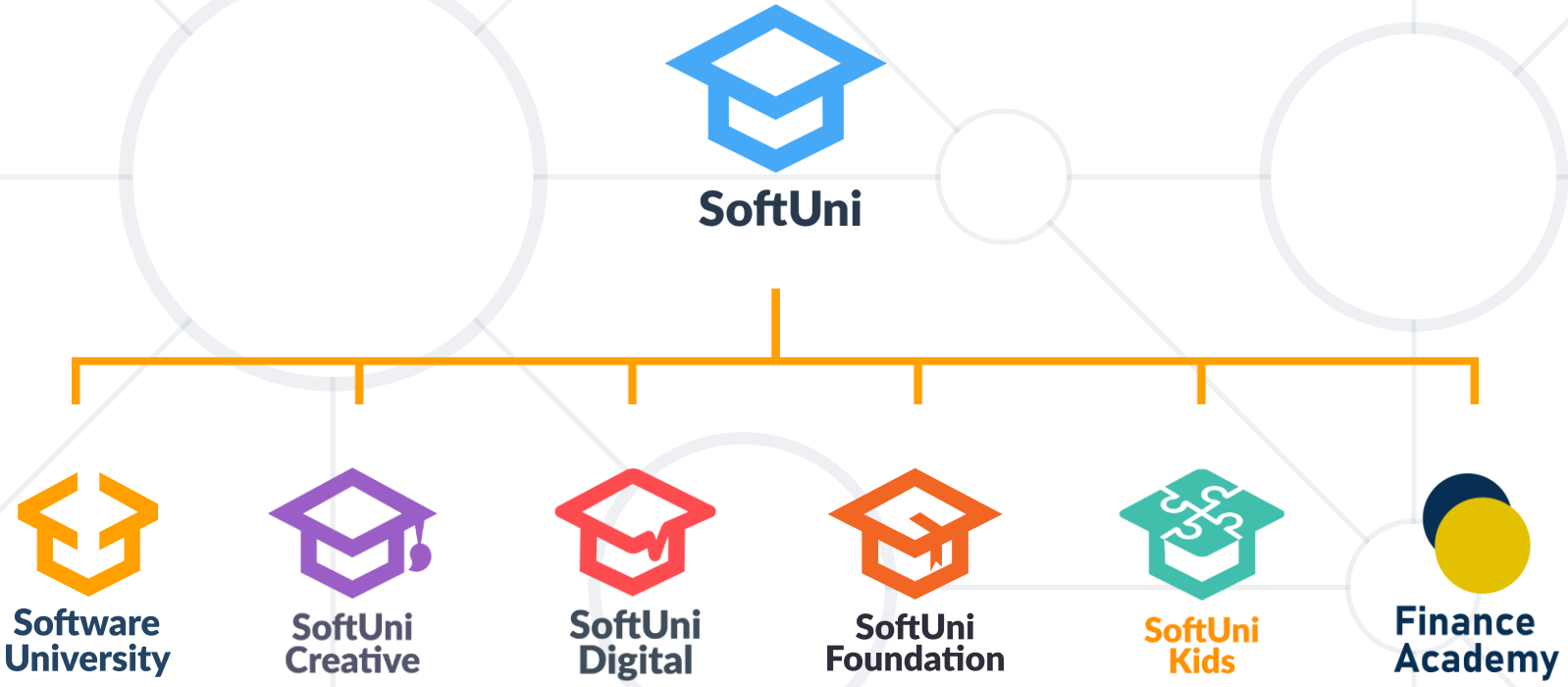
Will pass all messages to the console handler

Will pass all ERROR messages to the mail_admins handler

Summary

- Django raises some of its own exceptions as well as standard Python exceptions
- Handling exceptions helps to maintain the normal, desired flow of the program
- Logging provide you with more and better structured information about the state and health of your application

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

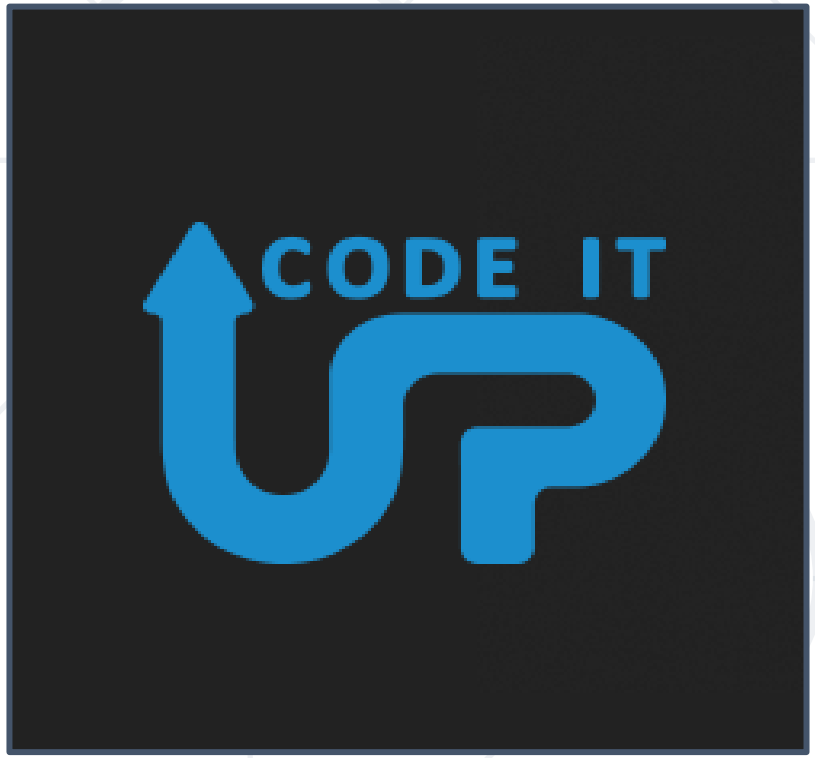
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg