Həftə 2 — Python Advanced Features: Tapşırıqlar

Gün 1 — File I/O, Exception Handling

Mövzular: Fayldan oxuma/yazma, try/except/finally, raising exceptions

Assignment 1.1 — Əməliyyat Jurnalı Sistemi (Transaction Logger)

Bir maliyyə əməliyyat sisteminin jurnalını fayla yazan və oxuyan proqram yazın. Bütün xətalar düzgün idarə olunmalıdır.

Initial Data — `transactions_input.txt` faylı yaradın:
@transactions_input.txt

Account balansları (başlanğıc):

Tələblər:

1. **transactions_input.txt** faylını sətir-sətir oxuyun və hər əməliyyatı parse edin
2. **Custom Exception** sinifləri yaradın:
   - **InsufficientFundsError** — balans kifayət etmədikdə
   - **InvalidAccountError** — hesab mövcud olmadıqda
   - **InvalidTransactionError** — əməliyyat tipi tanınmadıqda
   - **DataCorruptionError** — məlumat formatı pozulduqda (məsələn, məbləğ rəqəm deyil)
3. Hər əməliyyat üçün uyğun exception-ları **raise** edin, **try/except** ilə tutun
4. Uğurlu əməliyyatları **transactions_success.txt**-ə, xətalıları isə **transactions_errors.txt**-ə yazın
5. **finally** blokunda fayl resurslarının bağlandığından əmin olun
6. Proqramın sonunda hesab balanslarını **balances_report.txt** faylına yazın
7. Xəta log faylında hər xəta üçün: tarix, xəta tipi, təfərrüat, orijinal sətir qeyd olunsun

Nümunə `transactions_errors.txt` çıxışı:

```
[2026-03-03] InsufficientFundsError: ACC-1003 hesabında 500.00 AZN var, 10000.00 AZN çıxarıla bilməz
  → Orijinal: 2026-03-03|WTH|ACC-1003|Tural Əliyev|10000.00|Alış
[2026-03-04] DataCorruptionError: Məbləğ rəqəm deyil: 'abc'
  → Orijinal: 2026-03-04|DEP|ACC-1004|INVALID_AMOUNT|abc|Maaş
[2026-03-04] InvalidAccountError: ACC-9999 hesabı mövcud deyil
  → Orijinal: 2026-03-04|WTH|ACC-9999|Naməlum|100.00|ATM
[2026-03-05] InvalidTransactionError: Naməlum əməliyyat tipi: 'XYZ'
  → Orijinal: 2026-03-05|XYZ|ACC-1001|Əli Həsənov|50.00|Test
```

Assignment 1.2 — Konfiqurasiya Faylı Parser

Strukturlaşdırılmış konfiqurasiya fayllarını oxuyan, validasiya edən, dəyişdirən və geri yazan sistem yazın. **.ini** formatında olan konfiqurasiya fayllarını manual parse edin (hazır **configparser** istifadə etməyin).

Initial Data — `server_config.ini` faylı yaradın:
@server_config.ini

Tələblər:

1. **parse_config(filepath) -> dict** — faylı oxuyub nested dict qaytarsın: **{"server": {"host": "...", ...}, "database": {...}}**
2. **validate_config(config) -> list[str]** — aşağıdakı qaydaları yoxlasın:
   - **port** dəyərləri 1-65535 aralığında olmalı
   - **host** boş olmamalı
   - **password** minimum 8 simvol olmalı
   - **pool_size** müsbət tam ədəd olmalı
   - **level** yalnız **DEBUG**, **INFO**, **WARNING**, **ERROR**, **CRITICAL** ola bilər
   - Validasiya xətalarını list olaraq qaytarsın
3. **get_value(config, section, key, default=None)** — dəyəri qaytarsın, mövcud deyilsə **default**, tip çevirməsini (**int**, **bool**) avtomatik etsin
4. **set_value(config, section, key, value)** — dəyəri dəyişsin, section mövcud deyilsə yaratsın
5. **write_config(config, filepath)** — config-i orijinal formatda fayla yazsın
6. Bütün fayl əməliyyatlarını **try/except/finally** ilə qoruyun
7. Pozulmuş fayllar (**]** olmayan section, **=** olmayan sətir, Unicode xətaları) üçün uyğun exception-lar raise edin
8. **diff_configs(config1, config2) -> dict** — iki config arasındakı fərqləri tapıb qaytarsın

Assignment 1.3 — CSV Data Prosessoru

Böyük CSV fayllarını emal edən, filtrləyən, transformasiya edən və nəticəni yeni fayla yazan proqram yazın. Hazır **csv** modulu yalnız ayrıştırma üçün istifadə edə bilərsiniz, analiz məntiqi tamamilə öz kodunuz olmalıdır.

Initial Data — `employees.csv` faylı yaradın:
@employees.csv

Tələblər:

1. CSV faylını oxuyun, hər sətri dict olaraq parse edin (headers → keys)
2. Oxuma zamanı baş verə biləcək xətaları idarə edin:
   - Fayl mövcud deyilsə → **FileNotFoundError**
   - Sətirdə gözlənilən sütun sayı düz deyilsə → **DataCorruptionError** (custom)
   - **salary** və ya **performance_score** rəqəm deyilsə → **ValueError** tutun, log-layın, həmin sətri atlayın
3. Filtrləmə funksiyaları:
   - **filter_by_department(data, dept)** — verilmiş departament üzrə
   - **filter_by_salary_range(data, min_sal, max_sal)** — maaş aralığına görə
   - **filter_by_performance(data, min_score)** — performans skoruna görə
4. Transformasiya:
   - Hər işçi üçün **əmək stajını** hesablayın (illərlə, bugünkü tarixə əsasən)
   - Performansa görə bonus hesablayın: 90+ → 15%, 80-89 → 10%, 70-79 → 5%, <70 → 0%

- Yeni sütunlar əlavə edin: **experience_years**, **bonus_amount**, **total_compensation**

5. Nəticələri **employees_processed.csv** faylına yazın
6. Departament üzrə statistik hesabat yaradın → **department_report.txt**
7. Xəta jurnalını **processing_errors.log** faylına yazın
8. Bütün fayl əməliyyatlarında **with** statement istifadə edin

Bonus Assignment 1.B — Refaktor: Modul Strukturu ilə Yenidən Qurma

> *Bu tapşırığı Gün 2-nin mövzularını öyrəndikdən sonra edin.*

Assignment 1.3 (CSV Prosessor) kodunu aşağıdakı paket strukturuna refaktor edin:

```
csv_processor/
 __init__.py
 reader.py        # CSV oxuma funksiyaları
 writer.py        # CSV və hesabat yazma funksiyaları
 filters.py       # Filtrləmə funksiyaları
 transforms.py    # Transformasiya funksiyaları
 exceptions.py    # Custom exception sinifləri
 validators.py    # Data validasiya funksiyaları
 main.py          # Əsas proqram
```

1. Hər moduldakı funksiyaları uyğun fayla köçürün
2. **__init__.py**-da əsas funksiyaları **import** edin ki, **from csv_processor import read_csv, filter_by_department** kimi istifadə etmək mümkün olsun
3. **os.path** ilə fayl yollarını platforma-müstəqil edin
4. **sys.argv** ilə command-line argument qəbul edin: **python -m csv_processor --input data.csv --department Mühəndislik**

Gün 2 — Modules and Packages, Standard Library

Mövzular: Modul yaratma/import, paketlər, **__init__.py**, os, sys, math, datetime, json, collections, itertools

Assignment 2.1 — Fayl Sistem Analizatoru

os, **sys**, **math**, **datetime** modullarından istifadə edərək, verilmiş direktoriya ağacını analiz edən paket yazın.

Paket strukturu:

```
fs_analyzer/
 __init__.py
 scanner.py       # Direktoriya skan funksiyaları
 analyzer.py      # Analiz və statistika
 reporter.py      # Hesabat generasiyası
 utils.py         # Yardımçı funksiyalar (ölçü formatı və s.)
 main.py          # Əsas giriş nöqtəsi
```

Tələblər:

`scanner.py`

1. **scan_directory(path) -> list[dict]** — **os.walk** ilə bütün faylları tapın, hər fayl üçün dict qaytarın:

```
      {"path": "...", "name": "...", "ext": ".py", "size": 1024, "modified": datetime(...), "is_hidden": False}
```

2. **os.path.getsize**, **os.path.getmtime**, **os.path.splitext** istifadə edin
3. İcazə xətalarını (**PermissionError**) tutub log-layın, keçin

`analyzer.py`

4. **get_extension_stats(files) -> dict** — hər fayl uzantısı üçün: fayl sayı, ümumi ölçü, orta ölçü
5. **find_duplicates(files) -> list[tuple]** — eyni ölçülü və eyni adlı faylları potensial duplikat olaraq tapın
6. **find_large_files(files, threshold_mb) -> list** — verilmiş ölçüdən böyük faylları tapın
7. **get_age_distribution(files) -> dict** — faylları yaşlarına görə qruplaşdırın: "Bu həftə", "Bu ay", "Bu il", "Köhnə"
8. **collections.Counter** ilə ən çox rast gəlinən 10 uzantını tapın

`reporter.py`

9. **generate_text_report(stats, output_path)** — analiz nəticəsini formatlanmış tekst faylına yazsın
10. **generate_json_report(stats, output_path)** — **json** modulu ilə JSON formatında hesabat yazsın

`utils.py`

11. **format_size(bytes)** — baytı KB/MB/GB-ə çevirsin (**math.log** istifadə etməklə)
12. **format_date(timestamp)** — **datetime** ilə oxunaqlı tarix formatı

`main.py`

13. **sys.argv** ilə direktoriya yolunu və filtr parametrlərini qəbul etsin
14. **if __name__ == "__main__"** bloku ilə çalışsın

Assignment 2.2 — Riyazi Emal Paketi (Math Processing Toolkit)

math, **statistics**, **random**, **collections**, **itertools**, **functools** modullarından istifadə edərək, riyazi data emal paketi yazın.

Paket strukturu:

```
math_toolkit/
 __init__.py
 stats.py         # Statistik funksiyalar
 combinatorics.py # Kombinatorika
 sequences.py     # Ardıcıllıqlar
 utils.py         # Formatlama, validasiya
 main.py
```

Initial Data:

_Lazımi məlumatlar **assignment_2_2.py** faylında qeyd olunmuşdur._

Tələblər:

`stats.py`

1. **descriptive_stats(data) -> dict** — orta, median, moda, standart sapma, dispersiya, min, max, aralıq, kvartillər (**statistics** modulu)
2. **detect_outliers(data, threshold=2) -> list** — z-score metodu ilə outlier-ləri tapın (**math.sqrt** ilə manual hesablama, **statistics** ilə müqayisə)
3. **correlation(data_x, data_y) -> float** — Pearson korrelyasiya əmsalını hesablayın (manual formul + **math**)
4. **moving_average(data, window) -> list** — sürüşən orta hesablayın (**collections.deque** ilə)

`combinatorics.py`

5. **all_combinations(groups, size) -> list** — **itertools.combinations** ilə bütün mümkün qrup birləşmələrini tapın
6. **all_permutations(items) -> list** — **itertools.permutations**
7. **cartesian_product(*iterables)** — **itertools.product** ilə dekart hasili
8. **power_set(items)** — bir çoxluğun bütün alt çoxluqlarını tapın (**itertools.chain**, **combinations**)

`sequences.py`

9. **fibonacci_gen(n)** — generator ilə Fibonacci ardıcıllığı
10. **prime_sieve(limit)** — **math.isqrt** istifadə edərək sadə ədədləri tapın (Eratosfen ələyi)
11. **geometric_series(a, r, n)** — həndəsi silsilə (**math.pow**)
12. **collatz_sequence(n)** — Collatz ardıcıllığını generasiya edin

`main.py`

13. Bütün sensor datalarını emal edin, hər sensor üçün detallı hesabat çıxarın
14. Sensor-lar arası korrelyasiyanı hesablayın
15. Nəticəni **json** ilə fayla yazın
16. **functools.reduce** ilə bütün sensor datalarını birləşdirib ümumi statistika çıxarın

Assignment 2.3 — Tapşırıq İdarəetmə Sistemi (Task Management CLI)

datetime, **json**, **os**, **sys**, **collections** modullarından istifadə edərək, CLI əsaslı tapşırıq idarəetmə sistemi yazın. Datalar JSON faylda saxlanılsın.

Paket strukturu:

```
task_manager/
 __init__.py
 models.py        # Tapşırıq data modelləri
 storage.py       # JSON fayl əməliyyatları
 commands.py      # CLI əmr funksiyaları
 display.py       # Formatlanmış çıxış
```

```
validators.py    # Giriş validasiyası
main.py
```

Initial Data — `tasks.json` faylı yaradın:
@tasks.json

Tələblər:

`storage.py`

1. **load_data(filepath) -> dict** — JSON faylı oxusun, fayl mövcud deyilsə boş struktur qaytarsın
2. **save_data(data, filepath)** — datanı gözəl formatda (**json.dumps** ilə **indent**, **ensure_ascii=False**) JSON-a yazsın
3. **backup_data(filepath)** — saxlamadan əvvəl əvvəlki versiyanı **.bak** faylına kopyalasın (**os.rename**)

`models.py`

4. Tapşırıq yaratma, yeniləmə, silmə funksiyaları — **datetime.now()** ilə timestamp əlavə edin
5. ID generasiya: **T-{next_number}** formatında, **collections.defaultdict** ilə idarə edin

`commands.py`

6. **add_task(project_id, title, priority, assignee, tags)** — yeni tapşırıq əlavə etsin
7. **update_status(task_id, new_status)** — statusu yeniləsin (**todo** → **in_progress** → **done**)
8. **assign_task(task_id, assignee)** — tapşırığı komanda üzvünə təyin etsin
9. **list_tasks(project_id, **filters)** — filtrlərlə (status, priority, assignee, tag) siyahı
10. **search_tasks(keyword)** — başlıq və taglarda axtarış

`display.py`

11. Tapşırıqları gözəl formatda göstərin:

```
PRJ-001: Veb Sayt Redizayn  Deadline: 2026-04-15 (29 gün qalıb)

  T-001 [HIGH] Wireframe hazırla                  @Əli    #design
  T-002 [MED]  Rəng palitrasını seç               @Leyla  #design
  T-003 [HIGH] Responsive layout kodla            @Tural  #frontend
  T-004 [HIGH] API endpoint-ləri yaz              @Əli    #backend
  T-005 [MED]  Unit testlər yaz                   –       #testing

 Progress:  40% (2/5)
```

12. Komanda üzvü üzrə yüklənmə hesabatı: hər üzvün neçə aktiv tapşırığı var
13. Gecikmiş deadlineler üçün xəbərdarlıq (**datetime** ilə müqayisə)

`main.py`

14. **sys.argv** ilə CLI əmrləri qəbul etsin:

```
python -m task_manager add PRJ-001 "Yeni tapşırıq" --priority high --assignee Əli --tags backend,api
```

```
python -m task_manager list PRJ-001 --status todo --priority high
python -m task_manager status T-003 done
python -m task_manager report
```

Bonus Assignment 2.B — Refaktor: API İnteqrasiyası

> *Bu tapşırığı Gün 3-ün mövzularını öyrəndikdən sonra edin.*

Assignment 2.3 (Task Manager) sistemini genişləndirin:

1. Tapşırıqları **GitHub Issues** ilə sinxronlaşdırın: **requests** ilə GitHub API-dən issue-ları çəkin və yerli task-lara çevirin
2. Tapşırıq yaradıldıqda və ya tamamlandıqda **webhook** göndərin (POST request simulyasiyası)
3. Komanda üzvü haqqında məlumatı **web scraping** ilə GitHub profillərindən çəkin
4. Bütün API sorğularını **try/except** ilə qoruyun, əlaqə xətalarını log-layın

Gün 3 — Working with APIs, Web Scraping

Mövzular: HTTP requests, API əlaqəsi, JSON response, BeautifulSoup, web scraping

Assignment 3.1 — Multi-API Dashboard: NASA + GitHub + OpenRouter

Üç fərqli API-dən məlumat çəkib birləşdirən və analiz edən proqram yazın.

API Məlumatları:

API	Base URL	Auth
NASA	`https://api.nasa.gov`	API Key (pulsuz: `DEMO_KEY` və ya api.nasa.gov-da
GitHub	`https://api.github.com`	Token (opsional, limiti artırmaq üçün)
OpenRouter	`https://openrouter.ai/api/v1`	API Key (openrouter.ai-dan əldə edin)

Paket strukturu:

```
api_dashboard/
 __init__.py
 clients/
    __init__.py
    nasa_client.py
    github_client.py
    openrouter_client.py
 processors/
    __init__.py
    data_processor.py
 exceptions.py
 config.py         # API key-lərin idarəsi
 utils.py
 main.py
```

Tələblər:

`config.py`

1. API key-ləri environment variable-lardan oxuyun (**os.environ.get**), mövcud deyilsə **.env** faylından oxuyun
2. Key olmadıqda düzgün xəbərdarlıq verin

`exceptions.py`

3. Custom exception-lar: **APIConnectionError**, **RateLimitError**, **InvalidResponseError**, **AuthenticationError**

`nasa_client.py` — NASA API

4. **APOD** (Astronomy Picture of the Day): **GET /planetary/apod** — günün astronomiya şəklini alın
5. **Mars Rover Photos**: **GET /mars-photos/api/v1/rovers/curiosity/photos?sol=1000** — Mars fotoları
6. **NEO** (Near Earth Objects): **GET /neo/rest/v1/feed?start_date=2026-03-10&end_date=2026-03-17** — yaxın keçən asteroidlər
7. Hər endpoint üçün response-u parse edin, lazımi məlumatları çıxarın
8. Rate limit-ə uyğun retry mexanizmi əlavə edin

`github_client.py` — GitHub API

9. **User Info**: **GET /users/{username}** — istifadəçi profili
10. **Repos**: **GET /users/{username}/repos?sort=stars&per_page=10** — ən populyar repolar
11. **Languages**: **GET /repos/{owner}/{repo}/languages** — repo dillərinin statistikası
12. Pagination ilə işləyin (**Link** header parse)
13. Response header-dən **X-RateLimit-Remaining** yoxlayın

`openrouter_client.py` — OpenRouter API

14. **POST /chat/completions** — NASA datası haqqında süni intellektdən analiz alın:
15. Cavabı parse edin, xəta hallarını idarə edin

`data_processor.py`

16. NASA APOD + OpenRouter analizi birləşdirin
17. GitHub repo statistikası ilə NASA API response müqayisəsi (cavab vaxtı, data ölçüsü)
18. Bütün nəticələri JSON faylına yazın

`main.py`

19. Bütün API çağırışlarını ardıcıl icra edin
20. Hər API çağırışını **try/except** ilə qoruyun — bir API işləmirsə digərləri davam etsin
21. Nəticəni gözəl formatda çap edin:

```
            MULTI-API DASHBOARD — 2026-03-17
```

```
NASA — Günün Astronomiya Şəkli
  Başlıq: Spiral Galaxy NGC 1300
  Tarix: 2026-03-17
  Təsvir: A barred spiral galaxy located...
   AI Analiz: Bu qalaktika Eridanus bürcündə...

NASA — Yaxın Keçən Asteroidlər (bu həftə)
  Ümumi: 12 asteroid
  Potensial təhlükəli: 2
  Ən böyüyü: 2026-QR1 (450m diametr)

GitHub — İstifadəçi: torvalds
  Repolar: 7 | Follower: 200K+
  Ən populyar: linux ( 185K)
  Əsas dillər: C (85%), Assembly (10%)
```

Assignment 3.2 — Xəbər Aqreqatoru (Web Scraper)

Müxtəlif xəbər mənbələrindən məlumat toplayan web scraper yazın.

Hədəf saytlar (yalnız publik, robots.txt-ə uyğun saytlar):

Mənbə	URL	Çıxarılacaq data
Hacker News	`https://news.ycombinator.com`	Başlıqlar, xallar, müəlliflər, linklər
Python.org	`https://www.python.org/blogs/`	Blog başlıqları, tarixlər
GitHub Trending	`https://github.com/trending`	Trend olan repolar, dillər, ulduzlar

Paket strukturu:

```
news_scraper/
 __init__.py
 scrapers/
    __init__.py
    base_scraper.py      # Baza sinif (funksiyalar toplusu)
    hackernews_scraper.py
    python_blog_scraper.py
    github_trending_scraper.py
 exporters/
    __init__.py
    json_exporter.py
    csv_exporter.py
    text_exporter.py
 utils.py
 main.py
```

Tələblər:

`base_scraper.py`

1. Ümumi funksiyalar yazın:
   - **fetch_page(url, headers=None) -> str** — **requests.get** ilə səhifəni alın, status yoxlayın
   - **parse_html(html_content) -> BeautifulSoup** — HTML-i parse edin
   - **rate_limit_wait(seconds)** — sorğular arasında gözləmə

`hackernews_scraper.py`

2. Hacker News-un ana səhifəsindən ilk 30 xəbəri çıxarın:
   - Başlıq, link, xal sayı, müəllif, şərh sayı
3. Hər xəbəri dict olaraq qaytarın
4. Xəbərləri xal sayına görə sıralayın

`python_blog_scraper.py`

5. Python.org blog səhifəsindən blog yazılarını çıxarın:
   - Başlıq, tarix, qısa açıqlama, link
6. Tarixləri **datetime** obyektinə çevirin

`github_trending_scraper.py`

7. GitHub Trending səhifəsindən trend repoları çıxarın:
   - Repo adı, təsvir, proqramlaşdırma dili, ulduz sayı, bugünkü ulduzlar, fork sayı
8. Dilə görə filtrləmə dəstəkləyin

`exporters/`

9. Bütün nəticələri 3 formatda export edin: JSON, CSV, formatlanmış text

`main.py`

10. Bütün scraper-ləri ardıcıl icra edin
11. Hər scraper üçün **try/except** — bir sayt işləmirsə digərləri davam etsin
12. Nəticə hesabatı:

```
          XƏBƏR AQREQATORU — 2026-03-17


Hacker News — Top 10
 1. [358 ] Show HN: I built a... (author: pg)
 2. [291 ] The Future of... (author: dang)
 ...

Python.org Blog — Son Yazılar
 1. [2026-03-15] Python 3.14 Release...
 2. [2026-03-10] New PEP Accepted...
 ...
```

```
GitHub Trending — Top 5
 1.  1.2K today | awesome-project (Python)
 2.  890 today | rust-framework (Rust)
 ...

Ümumi Statistika
 Toplanan xəbər: 45
 Mənbə sayı: 3
 Export edildi: data/ (JSON, CSV, TXT)
```

Assignment 3.3 — API Müqayisə və Monitoring Aləti

Müxtəlif API-lərin performansını, etibarlılığını və data keyfiyyətini ölçən monitoring sistemi yazın.

Test ediləcək API-lər:

Tələblər:

1. Hər API-yə **3 dəfə** sorğu göndərin (ardıcıl) və hər sorğunun aşağıdakı metrikalarını toplayın:
   - Response time (ms)
   - Status code
   - Response body size (bytes)
   - Header-lər (Content-Type, rate limit info)
2. Hər API üçün orta response time, min/max, uğur faizi hesablayın
3. API-ləri response time-a görə sıralayın
4. Xəta hallarını kateqoriyalara ayırın: Connection Error, Timeout, 4xx, 5xx
5. Response body-ni validasiya edin: JSON parse olur mu, gözlənilən key-lər var mı
6. **Health check** funksiyası yazın — hər API-nin "sağlam" olub-olmadığını müəyyən edin:
   - Sağlam: response < 2s, status 200, JSON valid
   - Xəbərdarlıq: response 2-5s və ya bəzi sorğular uğursuz
   - Kritik: response > 5s və ya əksər sorğular uğursuz
7. Rate limit məlumatlarını header-lərdən parse edin (GitHub: **X-RateLimit-***, NASA: response-da)
8. Nəticəni JSON faylına yazın, formatlanmış hesabat çap edin:

```
          API MONİTORİNQ HESABATI — 2026-03-17


API                Orta (ms)  Min     Max     Uğur   Status

JSONPlaceholder      125        98     152    100%    Sağlam
GitHub Users         340       280     420    100%    Sağlam
GitHub Repos         380       310     450    100%    Sağlam
NASA APOD            890       650    1200    100%    Sağlam
```

```
OpenRouter Models   1500      1200    1900     67%    Xəbərdarlıq
HTTPBin             2200      1800    2800     33%    Kritik

 Ümumi Statistika
   Test edilmiş API: 6
   Ümumi sorğu: 18
   Uğurlu: 16 (89%)
   Orta response: 905 ms

 Ən sürətli: JSONPlaceholder (125 ms)
 Ən yavaş: HTTPBin (2200 ms)
```

Həftəlik Assignment — Şəxsi Maliyyə İdarəetmə Sistemi

> *Bu tapşırıq bütün həftənin mövzularını əhatə edir: File I/O, Exception Handling, Modules/Packages, Standard Library, APIs, Web Scraping.*

Sistemin Təsviri

Şəxsi maliyyə əməliyyatlarını izləyən, büdcə planlaşdırması edən, valyuta çevirən və maliyyə xəbərlərini toplayan tam funksional CLI sistemi yazın.

Paket strukturu:

```
finance_manager/
 __init__.py
 core/
    __init__.py
    transaction.py    # Əməliyyat əlavə/silmə/yeniləmə
    budget.py         # Büdcə planlaşdırması
    analytics.py      # Analiz və statistika
    exceptions.py     # Custom exceptions
 data/
    __init__.py
    file_handler.py   # CSV/JSON oxuma-yazma
    validators.py     # Data validasiyası
 external/
    __init__.py
    currency_api.py   # Valyuta API
    news_scraper.py   # Maliyyə xəbərləri
    ai_advisor.py     # OpenRouter ilə AI məsləhət
 display/
    __init__.py
    formatter.py      # Formatlanmış çıxış
 config.py
 main.py
```

Initial Data — `transactions.csv`:

@transactions.csv

Initial Data — `budget.json`:
@budget.json

Tələblər

Hissə 1 — Fayl Əməliyyatları və Exception Handling

1. **transactions.csv** faylını oxuyun, hər sətri dict-ə çevirin. Pozulmuş sətirlər üçün **DataCorruptionError** raise edin, log-layın, atlayın
2. Yeni əməliyyat daxil edin — faylın sonuna əlavə edin (**append** mode). Validasiya: tarix formatı, müsbət məbləğ, mövcud kateqoriya
3. **budget.json**-u oxuyub yazın, structure validasiyası edin
4. Custom exception-lar: **BudgetExceededError**, **InvalidTransactionError**, **CurrencyConversionError**, **DataCorruptionError**
5. Bütün fayl əməliyyatlarını **try/except/finally** ilə qoruyun
6. Hər gün sonunda avtomatik backup yaradın: **transactions_backup_2026-03-17.csv**

Hissə 2 — Modullar və Standard Library

7. **datetime** ilə: aylıq/həftəlik qruplaşdırma, tarixlər arası fərq, trend analizi
8. **json** ilə: konfiqurasiya oxuma/yazma, API response emalı
9. **os** ilə: backup fayl idarəsi, direktoriya yaratma, fayl mövcudluğu yoxlaması
10. **collections.Counter** ilə: ən çox xərclənən kateqoriyalar
11. **collections.defaultdict** ilə: kateqoriyaya görə qruplaşdırma
12. **math**/**statistics** ilə: orta xərc, standart sapma, trend hesablaması
13. **sys.argv** ilə CLI əmrləri:

```
python -m finance_manager add --type expense --category Qida --amount 50 --desc "Nahar"
python -m finance_manager report --month 2026-03
python -m finance_manager budget --check
python -m finance_manager convert 100 USD AZN
```

Hissə 3 — API və Web Scraping

14. **Valyuta çevirməsi** — pulsuz API ilə (**https://api.exchangerate-api.com/v4/latest/USD** və ya oxşar) canlı məzənnə alın, USD əməliyyatları AZN-ə çevirin
15. **Maliyyə xəbərləri** — web scraping ilə maliyyə xəbərlərini çəkin (məsələn, **https://news.ycombinator.com** finance tag-ı, və ya digər)
16. **AI Maliyyə Məsləhətçisi** — OpenRouter API ilə xərcləmə vərdişinizə əsasən AI-dan maliyyə tövsiyəsi alın:

```
    prompt = f"""Son ayda xərclərim:
    {category_summary}
```

```
    Büdcə limitim: {budget_limits}
    Maliyyə məsləhəti ver: hansi kateqoriyalarda qənaət edə bilərəm?"""
```

17. Bütün API çağırışlarını **try/except** ilə qoruyun; API işləmədikdə offline rejimə keçin

Nəticə Hesabatı

18. Sistem aşağıdakı hesabatı çıxara bilməlidir:

```
        MALİYYƏ HESABATI — Mart 2026


 Ümumi Göstəricilər
   Gəlir:  3,450.00 AZN (650.00 USD → 1,105.00 AZN)
   Xərc:   1,618.00 AZN
   Qənaət: 1,832.00 AZN (Hədəf: 500 AZN )

 Kateqoriya Üzrə Xərclər  Büdcə Müqayisəsi
Kateqoriya     Xərclənib  Limit    Qalıq    Status

Kirayə          800.00     800.00     0.00   100%
Qida            335.00     500.00   165.00    67%
Nəqliyyat        48.00     100.00    52.00    48%
Kommunal         80.00     150.00    70.00    53%
Əyləncə          25.00     200.00   175.00    13%
Geyim           180.00     250.00    70.00    72%
Texnologiya      95.00     200.00   105.00    48%
Sağlamlıq        55.00     100.00    45.00    55%

 Valyuta Məzənnəsi
   1 USD = 1.70 AZN (canlı məzənnə)
   USD gəlir: 650.00 $ → 1,105.00 ₼

  Maliyyə Xəbərləri
  1. [2026-03-17] AZN/USD exchange rate holds steady...
  2. [2026-03-16] New savings regulation announced...

  AI Maliyyə Məsləhəti
  Xərc vərdişinizə əsasən: Qida kateqoriyasında...
  Qənaət potensialı: aylıq ~150 AZN əlavə qənaət...
```