Həftə 5 — Code Quality, Testing, Debugging: Tapşırıqlar

Gün 1 — Code Quality (Clean Code, Code Reviews)

Mövzular: Readability, maintainability, simplicity, code review best practices

Assignment 1.1 — Kod Refaktorinqi: "Spaghetti"dən Təmiz Koda

Aşağıdakı bilərəkdən pis yazılmış kodu clean code prinsiplərinə uyğun refaktor edin.

Refaktor ediləcək kod:

```
def p(d,t):
 r=[]
 for i in d:
  if t=="a":
   if i["b"]>1000 and i["s"]=="active":
    x=i["b"]*0.05
    if i["ty"]=="premium":
     x=i["b"]*0.08
    i["b"]=i["b"]+x
    r.append(i)
  elif t=="b":
   if i["b"]<0:
    print(f"WARNING: {i['n']} has negative balance: {i['b']}")
    i["f"]=True
    r.append(i)
  elif t=="c":
   total=0
   for j in i["tr"]:
    if j["type"]=="credit":
     total+=j["amount"]
    elif j["type"]=="debit":
     total-=j["amount"]
   i["calc_balance"]=total
   if abs(total-i["b"])>0.01:
    i["discrepancy"]=True
    r.append(i)
 return r

def gen_rep(accounts,start,end):
 res={"total_accounts":0,"total_balance":0,"premium":0,"standard":0,"flagged":0,"avg":0}
 fl=[]
 for a in accounts:
  if a.get("created"):
   from datetime import datetime
   d=datetime.strptime(a["created"],"%Y-%m-%d")
   if d>=datetime.strptime(start,"%Y-%m-%d") and d<=datetime.strptime(end,"%Y-%m-%d"):
    res["total_accounts"]+=1
    res["total_balance"]+=a["b"]
    if a["ty"]=="premium":res["premium"]+=1
    else:res["standard"]+=1
    if a.get("f"):res["flagged"]+=1;fl.append(a["n"])
 if res["total_accounts"]>0:res["avg"]=res["total_balance"]/res["total_accounts"]
 res["flagged_names"]=fl
 return res

def val(email,phone,date):
 import re
 errors=[]
 if not re.match(r"[^@]+@[^@]+\.[^@]+",email):errors.append("bad email")
 if not re.match(r"^\+?[0-9]{10,15}$",phone):errors.append("bad phone")
 try:
  from datetime import datetime
  datetime.strptime(date,"%Y-%m-%d")
 except:errors.append("bad date")
 if len(errors)>0:return False,errors
 return True,[]

DATA=[

{"n":"Eli","b":5000,"s":"active","ty":"premium","f":False,"created":"2025-01-15","tr":[{"type":"credit","amount":5000},{"type"
```

```
:"debit","amount":200}]},

{"n":"Leyla","b":1200,"s":"active","ty":"standard","f":False,"created":"2025-03-20","tr":[{"type":"credit","amount":1500},{"ty
pe":"debit","amount":300}]},

{"n":"Tural","b":-50,"s":"active","ty":"standard","f":False,"created":"2025-06-10","tr":[{"type":"credit","amount":100},{"type
":"debit","amount":150}]},

{"n":"Nigar","b":8500,"s":"inactive","ty":"premium","f":False,"created":"2025-02-28","tr":[{"type":"credit","amount":9000},{"t
ype":"debit","amount":500}]},

{"n":"Rashad","b":3200,"s":"active","ty":"premium","f":False,"created":"2025-08-05","tr":[{"type":"credit","amount":3000},{"ty
pe":"debit","amount":100}]},
]
```


Tələblər:

1. **Adlandırma**: bütün dəyişən, funksiya, parametr adlarını mənalı edin
2. **Funksiya parçalanması**: böyük funksiyaları kiçik, tək məsuliyyətli funksiyalara bölün (SRP)
3. **Magic number/string yoxdur**: sabitlər üçün `CONSTANTS` istifadə edin
4. **Type hints** əlavə edin: `def calculate_interest(account: dict, rate: float) -> float`
5. **Docstring** yazın: hər funksiyaya Google style docstring
6. **Import-lar üst hissədə** olsun, funksiya daxilində yox
7. **Dictionary key-lərini** mənalı edin: `"b"` → `"balance"`, `"n"` → `"name"` və s.
8. **Error handling** düzəldin: çılpaq `except:` yox, spesifik exception-lar
9. Refaktordan sonra funksionallıq eyni qalmalıdır (əvvəl/sonra eyni nəticə)
10. Code review checklist yaradın: bu kodda nələr səhv idi (minimum 15 maddə)

Assignment 1.2 — Clean Code Kitabxanası Yaratma

Sıfırdan clean code prinsiplərinə riayət edərək URL qısaldıcı kitabxanası yazın.

Tələblər:

1. Butün kodda aşağıdakı prinsiplərə riayət edin və hər birini tətbiq etdiyiniz yeri şərh edin:
   - **SRP** (Single Responsibility) — hər funksiya/modul bir iş görsün
   - **DRY** (Don't Repeat Yourself) — təkrar kod olmasın
   - **KISS** (Keep It Simple) — lazımsız mürəkkəblik olmasın
   - **YAGNI** (You Ain't Gonna Need It) — lazım olmayan feature yazılmasın
1. Modul strukturu:

```
url_shortener/
 __init__.py
 shortener.py     # Əsas URL qısaldıcı məntiq
 storage.py       # URL saxlama (dict/file)
 validators.py    # URL validasiyası
 analytics.py     # Kliklər, statistika
 exceptions.py    # Custom exception-lar
 config.py        # Konfiqurasiya sabitləri
```

1. Hər funksiyada: type hints, docstrings, max 20 sətir
2. Naming convention: `snake_case` funksiyalar, `PascalCase` siniflər, `UPPER_CASE` sabitlər

3. Funksionallıq: URL qısalt, orijinalı qaytar, klik say, statistika, müddətli link (expire), custom alias
4. Heç bir `print()` statement olmasın — `logging` modulu istifadə edin

Assignment 1.3 — Kod Review Simulyatoru

Başqasının yazdığı kodu review edib, detallı review report yazın.

Review ediləcək kod:

```
import json, os, sys, time, random
from datetime import datetime, timedelta

class TaskManager:
    def __init__(self):
        self.tasks = []
        self.file = "tasks.json"
        self.load()

    def load(self):
        try:
            f = open(self.file, "r")
            self.tasks = json.load(f)
            f.close()
        except:
            self.tasks = []

    def save(self):
        f = open(self.file, "w")
        json.dump(self.tasks, f)
        f.close()

    def add(self, title, desc="", pri=1, assign=None, due=None):
        id = len(self.tasks) + 1  # BUG: silindikdən sonra ID təkrarlana bilər
        task = {"id": id, "title": title, "description": desc, "priority": pri,
                "assignee": assign, "due_date": due, "status": "todo",
                "created": str(datetime.now()), "subtasks": [], "tags": []}
        self.tasks.append(task)
        self.save()
        return task

    def delete(self, id):
        for i in range(len(self.tasks)):
            if self.tasks[i]["id"] == id:
                del self.tasks[i]
                self.save()
                return True
        return False

    def update_status(self, id, status):
        # status validasiyası yoxdur
        for t in self.tasks:
            if t["id"] == id:
                t["status"] = status
                self.save()
                return True
        return False

    def search(self, keyword):
        results = []
        for t in self.tasks:
            if keyword.lower() in t["title"].lower() or keyword.lower() in t["description"].lower():
                results.append(t)
        return results

    def get_overdue(self):
        overdue = []
        for t in self.tasks:
            if t["due_date"] and t["status"] != "done":
                # String müqayisəsi — düzgün deyil
```

```
            if t["due_date"] < str(datetime.now()):
                overdue.append(t)
        return overdue

    def get_stats(self):
        stats = {}
        stats["total"] = len(self.tasks)
        stats["todo"] = len([t for t in self.tasks if t["status"] == "todo"])
        stats["in_progress"] = len([t for t in self.tasks if t["status"] == "in_progress"])
        stats["done"] = len([t for t in self.tasks if t["status"] == "done"])
        stats["overdue"] = len(self.get_overdue())
        if len(self.tasks) > 0:
            stats["completion_rate"] = stats["done"] / stats["total"] * 100
        else:
            stats["completion_rate"] = 0
        return stats

    def assign_random(self, team):
        # Yük balanslaması olmadan təsadüfi təyin edir
        for t in self.tasks:
            if t["assignee"] is None:
                t["assignee"] = random.choice(team)
        self.save()

    def export_csv(self, filename):
        f = open(filename, "w")
        f.write("id,title,status,priority,assignee,due_date\n")
        for t in self.tasks:
            line = f"{t['id']},{t['title']},{t['status']},{t['priority']},{t['assignee']},{t['due_date']}\n"
            f.write(line)
        f.close()

    def bulk_update(self, ids, field, value):
        count = 0
        for t in self.tasks:
            if t["id"] in ids:
                t[field] = value  # Təhlükəsiz deyil — istənilən sahəni dəyişə bilər
                count += 1
        self.save()
        return count
```

Tələblər:

1. Kodda olan bütün problemləri tapın, kateqoriyalara bölün:
   - **Bug** — proqramın düzgün işləməsinə mane olan xətalar
   - **Code Smell** — işləyir amma pis praktikadır
   - **Improvement** — daha yaxşı edilə bilər
   - **Security** — təhlükəsizlik problemi
   - **Performance** — performans problemi
1. Hər problem üçün: harada (sətir nömrəsi), nə problem, niyə problem, necə düzəltmək
2. Minimum **20 problem** tapın
3. Düzəldilmiş versiyasını yazın — bütün tapılan problemlər həll olunmuş
4. Code review report-u markdown formatında yazın

Bonus Assignment 1.B — Refaktor: Test Coverage ilə Doğrulama

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

Assignment 1.1-dən refaktor edilmiş kodu `pytest` ilə test edin:

1. Refaktordan əvvəlki və sonrakı nəticələrin eyni olduğunu test edin
2. Hər yeni funksiya üçün minimum 3 test yazın
3. Edge case-ləri test edin: boş data, mənfi balans, yanlış tarix

---

Gün 2 — Unit Testing (unittest, pytest, TDD)

Mövzular: unittest, pytest frameworks, TDD workflow (Red-Green-Refactor)

Assignment 2.1 — Mövcud Koda Test Yazmaq

Aşağıdakı hesablama kitabxanasına hərtərəfli testlər yazın — həm `unittest` həm `pytest` ilə.

Test ediləcək kod:

```
# calculator.py
from datetime import datetime, timedelta
from typing import Union

class Calculator:
    def __init__(self):
        self.history: list[dict] = []
        self.memory: float = 0.0

    def add(self, a: float, b: float) -> float:
        result = a + b
        self._record("add", a, b, result)
        return result

    def subtract(self, a: float, b: float) -> float:
        result = a - b
        self._record("subtract", a, b, result)
        return result

    def multiply(self, a: float, b: float) -> float:
        result = a * b
        self._record("multiply", a, b, result)
        return result

    def divide(self, a: float, b: float) -> float:
        if b == 0:
            raise ZeroDivisionError("Sıfıra bölmək olmaz")
        result = a / b
        self._record("divide", a, b, result)
        return result

    def power(self, base: float, exp: float) -> float:
        if base == 0 and exp < 0:
            raise ValueError("Sıfırın mənfi qüvvəti mövcud deyil")
        result = base ** exp
        self._record("power", base, exp, result)
        return result

    def percentage(self, value: float, percent: float) -> float:
        result = value * percent / 100
        self._record("percentage", value, percent, result)
        return result

    def chain(self, initial: float, *operations: tuple) -> float:
        result = initial
        for op, value in operations:
            method = getattr(self, op, None)
            if method is None:
                raise AttributeError(f"Naməlum əməliyyat: {op}")
            result = method(result, value)
        return result

    def store_memory(self, value: float) -> None:
        self.memory = value

    def recall_memory(self) -> float:
        return self.memory

    def clear_history(self) -> None:
        self.history.clear()
```

```
    def get_history(self, last_n: int = None) -> list[dict]:
        if last_n:
            return self.history[-last_n:]
        return self.history.copy()

    def _record(self, op: str, a: float, b: float, result: float) -> None:
        self.history.append({
            "operation": op, "operand_a": a, "operand_b": b,
            "result": result, "timestamp": datetime.now().isoformat()
        })


class StatisticsCalculator:
    @staticmethod
    def mean(data: list[float]) -> float:
        if not data:
            raise ValueError("Boş siyahı üçün orta hesablana bilməz")
        return sum(data) / len(data)

    @staticmethod
    def median(data: list[float]) -> float:
        if not data:
            raise ValueError("Boş siyahı")
        sorted_data = sorted(data)
        n = len(sorted_data)
        if n % 2 == 0:
            return (sorted_data[n//2 - 1] + sorted_data[n//2]) / 2
        return sorted_data[n//2]

    @staticmethod
    def mode(data: list[float]) -> list[float]:
        if not data:
            raise ValueError("Boş siyahı")
        from collections import Counter
        counts = Counter(data)
        max_count = max(counts.values())
        return sorted([val for val, count in counts.items() if count == max_count])

    @staticmethod
    def std_dev(data: list[float]) -> float:
        if len(data) < 2:
            raise ValueError("Minimum 2 element lazımdır")
        mean = sum(data) / len(data)
        variance = sum((x - mean) ** 2 for x in data) / (len(data) - 1)
        return variance ** 0.5

    @staticmethod
    def percentile(data: list[float], p: float) -> float:
        if not 0 <= p <= 100:
            raise ValueError("Percentile 0-100 arasında olmalıdır")
        sorted_data = sorted(data)
        index = (p / 100) * (len(sorted_data) - 1)
        lower = int(index)
        upper = lower + 1
        if upper >= len(sorted_data):
            return sorted_data[-1]
        weight = index - lower
        return sorted_data[lower] * (1 - weight) + sorted_data[upper] * weight
```

Tələblər:

1. **unittest ilə** (`test_calculator_unittest.py`):
   - `setUp` / `tearDown` istifadə edin
   - Normal hallar (minimum 3 test hər metod üçün)
   - Edge cases: sıfır, mənfi ədədlər, çox böyük ədədlər, float dəqiqliyi
   - Exception testləri: `assertRaises`
   - `TestCalculator` və `TestStatisticsCalculator` ayrı test sinifləri
1. **pytest ilə** (`test_calculator_pytest.py`):
   - `@pytest.fixture` ilə Calculator instance
   - `@pytest.mark.parametrize` ilə çoxlu test data
   - `pytest.raises` ilə exception testləri
   - `@pytest.mark.slow` custom marker
   - `conftest.py` ilə paylaşılan fixture-lər
1. Hər iki framework ilə minimum **40 test** yazın
2. `pytest --cov` ilə coverage hesablayın, **90%+ coverage** hədəfləyin

---

Assignment 2.2 — TDD ilə Sıfırdan: Parola İdarəçisi

Əvvəlcə testləri yazın, sonra kodu. TDD-nin Red-Green-Refactor dövrünü izləyin.

Funksionallıq:

- `generate_password(length, uppercase, lowercase, digits, symbols) -> str`
- `check_strength(password) -> dict` — güc dərəcəsi: weak/medium/strong/very_strong, nisbətlər
- `hash_password(password) -> str` — SHA-256 hash
- `verify_password(password, hash) -> bool`
- `PasswordVault` sinfi: `add_entry(service, username, password)`, `get_entry(service)`, `delete_entry(service)`, `search(keyword)`, `export_encrypted(filepath, master_key)`, `import_encrypted(filepath, master_key)`

Tələblər:

1. **Addım 1 (RED)**: Hər funksiya üçün testləri əvvəlcə yazın. Testlər fail etməlidir:

```
def test_generate_password_length():
    pwd = generate_password(length=16)
    assert len(pwd) == 16

def test_generate_password_has_uppercase():
    pwd = generate_password(length=20, uppercase=True)
    assert any(c.isupper() for c in pwd)
# ... daha çox test
```

1. **Addım 2 (GREEN)**: Testləri keçəcək minimum kodu yazın
2. **Addım 3 (REFACTOR)**: Kodu təmizləyin, prinsiplər gözləyin
3. Hər TDD dövrünü sənədləşdirin — hansı test yazıldı, hansı kod əlavə edildi
4. Minimum **30 test** yazın (funksionallıq + edge case + exception)
5. Vault üçün: fayla yazma/oxuma, encryption simulyasiyası (base64 + XOR), axtarış

Assignment 2.3 — Integration Test Layihəsi

Əvvəlki həftələrdə yazdığınız hər hansı bir sistemi (və ya aşağıdakı kodu) integration testlərlə test edin.

Test ediləcək sistem (sadələşdirilmiş e-commerce):

```
# ecommerce.py
class Product:
    def __init__(self, sku, name, price, stock):
        self.sku, self.name, self.price, self.stock = sku, name, price, stock

    def is_available(self, qty=1): return self.stock >= qty
    def reduce_stock(self, qty):
        if qty > self.stock: raise ValueError("Stok kifayət etmir")
        self.stock -= qty
    def restock(self, qty): self.stock += qty

class Cart:
    def __init__(self):
        self.items = {}  # {sku: {"product": Product, "qty": int}}

    def add_item(self, product, qty=1):
        if not product.is_available(qty): raise ValueError(f"{product.name} stokda yoxdur")
        if product.sku in self.items: self.items[product.sku]["qty"] += qty
        else: self.items[product.sku] = {"product": product, "qty": qty}

    def remove_item(self, sku):
        if sku not in self.items: raise KeyError("Məhsul səbətdə yoxdur")
        del self.items[sku]

    def get_total(self):
        return sum(item["product"].price * item["qty"] for item in self.items.values())

    def clear(self): self.items.clear()

class Order:
    _counter = 0
    def __init__(self, cart, customer_name):
        Order._counter += 1
        self.order_id = f"ORD-{Order._counter:04d}"
        self.items = dict(cart.items)
        self.total = cart.get_total()
        self.customer = customer_name
        self.status = "pending"

    def process_payment(self, amount):
        if amount < self.total: raise ValueError("Ödəniş kifayət etmir")
        self.status = "paid"
        for item_data in self.items.values():
            item_data["product"].reduce_stock(item_data["qty"])
        self.change = amount - self.total
        return self.change

    def cancel(self):
        if self.status == "paid":
            for item_data in self.items.values():
                item_data["product"].restock(item_data["qty"])
        self.status = "cancelled"

    def ship(self):
        if self.status != "paid": raise ValueError("Ödənilməmiş sifariş göndərilə bilməz")
        self.status = "shipped"
```

Tələblər:

1. **Unit testlər**: hər sinif üçün ayrı (Product, Cart, Order)
2. **Integration testlər**: siniflərin birlikdə işləməsini test edin:
   - Product yarat → Cart-a əlavə et → Order yarat → Ödəniş et → Stok azalsın
   - Sifariş ləğv et → Stok geri qayıtsın
   - Eyni məhsulu iki müxtəlif cart-a əlavə et → stok konflikti
1. **Fixture-lər**: `@pytest.fixture` ilə test data setup
2. **Mock** istifadə edin: ödəniş prosesini mock-layın (`unittest.mock.patch`)
3. **Parametrize**: müxtəlif ssenariləri parametrik test edin
4. Minimum **35 test** (15 unit + 15 integration + 5 edge case)

Bonus Assignment 2.B — Refaktor: Logging və Debug Dəstəyi

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

Assignment 2.3 (E-commerce) koduna əlavə edin:

1. `logging` modulu ilə bütün əməliyyatları log-layın (DEBUG, INFO, WARNING, ERROR)
2. `pdb.set_trace()` ilə debug nöqtələri əlavə edin, testlər zamanı istifadə edin
3. Log çıxışlarını faylda test edin: doğru mesajların yazıldığını `caplog` (pytest) ilə yoxlayın

---

Gün 3 — Debugging Techniques and Tools

Mövzular: Debugging strategiyaları, IDE debugger, logging module, pdb

---

Assignment 3.1 — Bug Ovçusu: Buglı Kodu Düzəlt

Aşağıdakı kodda bilərəkdən gizlədilmiş **15 bug** var. Tapın, düzəldin, hər birini sənədləşdirin.

Buglı kod:

```
import logging
from datetime import datetime, timedelta
from collections import defaultdict

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class BankAccount:
    interest_rate = 0.05  # BUG 1: class variable — bütün hesablarda eyni dəyişir

    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance
        self.transactions = []
        self.created = datetime.now()
        logger.info(f"Hesab yaradıldı: {owner}")

    def deposit(self, amount):
        logger.debug(f"Deposit cəhdi: {amount}")
        if amount < 0:  # BUG 2: 0 deposit-ə icazə verir, amma 0 mənasızdır
            raise ValueError("Məbləğ müsbət olmalıdır")
        self.balance += amount
        self.transactions.append({"type": "deposit", "amount": amount, "date": datetime.now()})
        return self.balance

    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Məbləğ müsbət olmalıdır")
        if amount > self.balance:
            logger.warning(f"Kifayət qədər vəsait yoxdur: {self.balance} < {amount}")
            return False  # BUG 3: exception yerinə False qaytarır — xəta istifadəçiyə çatmır
        self.balance =- amount  # BUG 4: -= əvəzinə =- (unary minus assignment)
        self.transactions.append({"type": "withdraw", "amount": amount, "date": datetime.now()})
        return self.balance

    def transfer(self, other, amount):
        result = self.withdraw(amount)
        if result:  # BUG 5: withdraw 0 qaytarsa bu False olacaq (falsy), amma əməliyyat uğurlu ola bilər
            other.deposit(amount)
            logger.info(f"Transfer: {self.owner} -> {other.owner}: {amount}")
            return True
        return False

    def calculate_interest(self, months):
        interest = self.balance * self.interest_rate * months  # BUG 6: illik faizi aylığa çevirmir (/12)
        self.deposit(interest)
        return interest

    def get_statement(self, days=30):
        cutoff = datetime.now() - timedelta(days=days)
        # BUG 7: >= əvəzinə > istifadə edib, cutoff günündəki əməliyyatlar çıxmır
        recent = [t for t in self.transactions if t["date"] > cutoff]
        return recent

    def __str__(self):
        return f"Account({self.owner}, {self.balance})"  # BUG 8: balance formatlanmır, 2500.0000001 kimi görünə bilər

    def __eq__(self, other):
          return self.balance == other.balance  # BUG 9: owner yoxlamır, fərqli şəxslərin eyni balanslı hesabları "bərabər"
olacaq
```

```
class Bank:
    def __init__(self, name):
        self.name = name
        self.accounts = {}
        self.daily_limit = 10000

    def create_account(self, owner, initial_balance=0):
        # BUG 10: eyni adlı ikinci hesab açmağa cəhd etsə əvvəlkini əzir
        self.accounts[owner] = BankAccount(owner, initial_balance)
        return self.accounts[owner]

    def find_account(self, owner):
        return self.accounts[owner]  # BUG 11: KeyError atır, .get() istifadə etmir

    def total_deposits(self):
        total = 0
        for acc in self.accounts:  # BUG 12: accounts dict-dir, acc key olacaq (string), .values() lazımdır
            total += acc.balance
        return total

    def get_top_accounts(self, n=5):
           sorted_accs = sorted(self.accounts.values(), key=lambda a: a.balance)  # BUG 13: ascending sıralayır, descending
lazımdır
        return sorted_accs[:n]

    def process_batch_transfers(self, transfers):
        results = []
        for t in transfers:
            sender = self.find_account(t["from"])
            receiver = self.find_account(t["to"])
            # BUG 14: daily limit yoxlanmır
            success = sender.transfer(receiver, t["amount"])
            results.append({"transfer": t, "success": success})
        return results

    def generate_report(self):
        report = {
            "bank": self.name,
            "total_accounts": len(self.accounts),
            "total_deposits": self.total_deposits(),
            "generated_at": datetime.now()  # BUG 15: datetime JSON-a serialize olmur, str lazımdır
        }
        return report
```

Tələblər:

1. Hər bugu tapın, izah edin, düzəldin — aşağıdakı formatda sənədləşdirin:

```
Bug #4: Yanlış operator istifadəsi
Yer: BankAccount.withdraw(), sətir 34
Problem: `self.balance =- amount` — bu `self.balance = -amount` deməkdir
Nəticə: Balans həmişə mənfi olur
Düzəliş: `self.balance -= amount`
Debug metodu: print(self.balance) əlavə edib withdraw çağırdım, mənfi dəyər gördüm
```

1. Hər bug üçün hansı debug texnikası ilə tapdığınızı yazın: `print debug`, `pdb`, `logging`, `unit test`, `code review`

2. Düzəldilmiş kodu işlədin, bütün bugların həll edildiyini test edin

---

Assignment 3.2 — Logging Framework Qurulması

Peşəkar logging sistemi qurun — müxtəlif səviyyələr, handler-lər, formatter-lər.

Tələblər:

1. `logging` modulu ilə multi-handler sistem qurun:
   - `StreamHandler` — konsola (WARNING+ göstərsin)
   - `FileHandler` — `app.log` faylına (DEBUG+ yazsın)
   - `RotatingFileHandler` — `app_rotating.log` (hər 1MB-da yeni fayl, 5 backup)
   - `TimedRotatingFileHandler` — hər gün yeni fayl
1. Custom `Formatter` — JSON formatında log:

```
{"timestamp": "2026-03-17T22:00:00", "level": "ERROR", "module": "payment", "function": "process", "message": "Transaction failed", "extra": {"order_id": "ORD-501"}}
```

1. Custom log səviyyəsi əlavə edin: `AUDIT` (INFO ilə WARNING arası) — təhlükəsizlik əməliyyatları üçün
2. `logger.addFilter()` ilə filter: müəyyən modul adlarından gələn logları bloklama
3. Context manager `LogContext` yazın — əməliyyat kontekstini log-a əlavə etsin:

```
with LogContext(user="əli", action="payment"):
    logger.info("Ödəniş başladı")  # avtomatik user və action əlavə olunur
```

1. Log decorator: `@log_function_call` — funksiyanın daxil olduğu parametrləri, nəticəsini, vaxtını log-lasın
2. Test edin: `test_logging.py` — doğru mesajların doğru fayla yazıldığını yoxlayın

Test data:

```
test_scenarios = [
    {"level": "DEBUG", "msg": "Cache hit for key: user_123"},
    {"level": "INFO", "msg": "User login successful", "extra": {"user": "eli", "ip": "192.168.1.10"}},
    {"level": "AUDIT", "msg": "Password changed", "extra": {"user": "leyla"}},
    {"level": "WARNING", "msg": "Rate limit approaching", "extra": {"current": 95, "max": 100}},
    {"level": "ERROR", "msg": "Database connection failed", "extra": {"db": "main", "retry": 3}},
    {"level": "CRITICAL", "msg": "System memory exhausted", "extra": {"usage": "98%"}},
]
```

Assignment 3.3 — Debug Detective: Performans Analizi

Yavaş işləyən proqramı debug edin — bottleneck-ləri tapın, optimallaşdırın, nəticəni sənədləşdirin.

Yavaş kod:

```
import time, random

def slow_search(data, target):
    """Siyahıda axtarış — çox yavaş"""
    for i in range(len(data)):
        for j in range(len(data)):  # BUG: niyə iç-içə loop?
            if data[i] == target:
                return i
    return -1

def slow_remove_duplicates(data):
    """Təkrarları silir — O(n³)"""
    result = []
    for item in data:
        is_dup = False
        for existing in result:
            if item == existing:
                is_dup = True
                break
        if not is_dup:
            result.append(item)
    return result

def slow_sort(data):
    """Bubble sort — O(n²) hətta sıralanmış data üçün"""
    arr = data.copy()
    for i in range(len(arr)):
        for j in range(len(arr) - 1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

def slow_aggregate(records):
    """Qruplaşdırma — hər dəfə siyahıdan axtarır"""
    groups = {}
    for record in records:
        key = record["category"]
        if key not in groups:
            groups[key] = {"items": [], "total": 0, "count": 0}
        groups[key]["items"].append(record)
        groups[key]["count"] += 1
        groups[key]["total"] += record["value"]
        # Hər dəfə ən böyüyü tapır — əvəzinə sorted saxlamaq olar
        groups[key]["max"] = max(r["value"] for r in groups[key]["items"])
        groups[key]["min"] = min(r["value"] for r in groups[key]["items"])
    return groups

def slow_string_builder(words):
    """String birləşdirmə — O(n²) yaddaş"""
    result = ""
    for word in words:
        result = result + word + " "  # hər dəfə yeni string yaranır
        result = result.strip()
        result = result + " "
    return result.strip()

# Test data
data = [random.randint(1, 10000) for _ in range(5000)]
records = [{"id": i, "category": random.choice(["A","B","C","D","E"]), "value": random.uniform(10,1000)} for i in
range(10000)]
```

```
words = [f"word_{i}" for i in range(5000)]
```

Tələblər:

1. `cProfile` ilə hər funksiyanı profilləyin: hansı sətir ən çox vaxt alır
2. `time.perf_counter()` ilə əvvəl/sonra ölçün
3. `line_profiler` (əgər quraşdırıla bilirsə) və ya manual ölçmə ilə sətir-sətir analiz
4. Hər funksiyanın **Big-O** mürəkkəbliyini müəyyən edin
5. Optimal versiyasını yazın — hər funksiyanı **10x+ sürətləndirin**
6. Müqayilə cədvəli hazırlayın:

```
Funksiya            əvvəl (san)  Sonra (san)  Sürət artımı  əvvəl O()   Sonra O()

slow_search            8.50         0.001       8500x       O(n²)       O(n)/O(logn)
slow_remove_dups       3.20         0.01         320x       O(n²)       O(n)
slow_sort              5.10         0.02         255x       O(n²)       O(n logn)
slow_aggregate         4.80         0.05          96x       O(n·k)      O(n)
slow_string_builder    2.30         0.003        767x       O(n²)       O(n)
```

Həftəlik Assignment — Kod Keyfiyyəti Auditor Platforması

> *Bütün həftənin mövzularını əhatə edir: clean code, code review, unittest/pytest, TDD, debugging, logging.*

Sistemin Təsviri

Python kod fayllarını analiz edib, keyfiyyət hesabatı yaradan avtomatik audit platforması yazın.

Test ediləcək nümunə fayllar (platform bunları analiz edəcək):

```
# sample_bad_code.py — bilərəkdən pis kod
def f(x,y,z):
 if x>0:
  if y>0:
   if z>0:
    return x+y+z
   else:
    return x+y
  else:
   return x
 else:
  return 0

class c:
 def __init__(s,a,b):
  s.a=a;s.b=b
 def m(s):
  import math
  return math.sqrt(s.a**2+s.b**2)
```

```
# sample_good_code.py — yaxşı yazılmış kod
"""İki nöqtə arasında məsafə hesablayan modul."""
import math
from dataclasses import dataclass

@dataclass
class Point:
    """İki ölçülü koordinat nöqtəsi."""
    x: float
    y: float

    def distance_to(self, other: 'Point') -> float:
        """Bu nöqtədən digər nöqtəyə olan məsafəni hesablayır."""
        return math.sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2)
```

Paket strukturu:

```
code_auditor/
 src/
    code_auditor/
        __init__.py
        analyzers/
           __init__.py
           style_analyzer.py    # Naming, formatting yoxlaması
           complexity_analyzer.py # Cyclomatic complexity
           docstring_analyzer.py  # Docstring mövcudluğu/keyfiyyəti
           smell_detector.py     # Code smell-lər
        reporters/
           __init__.py
           text_reporter.py
           json_reporter.py
        utils/
           __init__.py
           file_reader.py
           ast_helpers.py       # Python AST ilə kod analizi
        logging_config.py
        exceptions.py
        cli.py
 tests/
    conftest.py
    test_style_analyzer.py
    test_complexity_analyzer.py
    test_docstring_analyzer.py
    test_smell_detector.py
    test_reporters.py
 samples/            # Test ediləcək nümunə fayllar
 pyproject.toml
 README.md
```

Tələblər

Hissə 1 — Analizatorlar (Clean Code tətbiqi)

1. **Style Analyzer** — `ast` modulu ilə Python faylını parse edin:
   - Dəyişən adları: `snake_case` yoxlaması, tək hərfli adlar (a, b, x)
   - Funksiya uzunluğu: 20 sətirdən uzun → xəbərdarlıq
   - Import-lar: fayl başında olub-olmadığı
   - Naming convention-lar: sinif `PascalCase`, sabit `UPPER_CASE`
1. **Complexity Analyzer**:
   - Cyclomatic complexity: if/for/while/except sayına görə
   - Nesting depth: 3-dən dərin → xəbərdarlıq
   - Parametr sayı: 5-dən çox → xəbərdarlıq
1. **Docstring Analyzer**:
   - Funksiya/siniflərdə docstring varmı
   - Docstring keyfiyyəti: parametrlər açıqlanıbmı, return tipi yazılıbmı
1. **Smell Detector**:
   - God class (çox metodu olan sinif)
   - Long method, duplicate code (oxşar struktur)
   - Magic numbers, dead code (istifadə olunmayan import/dəyişən)

Hissə 2 — Test Suite (TDD ilə)

1. **TDD iş axını ilə yazın**: əvvəlcə testlər, sonra analizatorlar
2. `pytest` + `pytest-cov` ilə minimum **90% coverage**
3. Fixture-lər: nümunə Python kod stringləri
4. Parametrize: müxtəlif kod nümunələri üçün testlər
5. Minimum **50 test** (hər analizator üçün 10+ test)

Hissə 3 — Debugging və Logging

1. Professional logging: hər analizator modulu üçün ayrı logger
2. JSON formatında log: fayl adı, tapılan problem, ciddilik
3. `cProfile` ilə analizatorların performansını profilləyin
4. Debug rejimi (`--debug` flag): ətraflı log, ara nəticələr

Nəticə Hesabatı

1. CLI: `python -m code_auditor analyze path/to/file.py --format text|json`

```
     KOD KEYFİYYƏTİ HESABATI
    Fayl: sample_bad_code.py
    Tarix: 2026-03-17 22:45


Ümumi Qiymət: 35/100 (Zəif )

 Stil Analizi
Sətir 1: Funksiya adı 'f' çox qısadır
Sətir 1: Parametrlər 'x,y,z' mənasız adlardır
Sətir 12: Sinif adı 'c' PascalCase deyil
Sətir 14: Import funksiya daxilindədir

 Mürəkkəblik
f(): Cyclomatic complexity = 4 (hədəf: ≤3)
f(): Nesting depth = 4 (hədəf: ≤3)

 Docstring
f(): docstring yoxdur
c sinfi: docstring yoxdur
c.m(): docstring yoxdur

 Code Smell
Sətir 13: Tək hərfli atribut adları (a, b)
Sətir 1-10: Nested if-lər — early return tövsiyə
```