Практичні завдання до лекції Функціональний Python

Модуль 2: Структури даних для ФП

Завдання: `namedtuple` (Незмінні дані)

Завдання: У вас є дані про студентів у вигляді простих кортежів. Це нечитабельно. Реорганізуйте дані, використовуючи `namedtuple`, для доступу до полів за іменем.

```
from collections import namedtuple

# Дано:
student_data = [("Володимир", "Іваненко", 20, "Київ"),
                ("Олена", "Петренко", 21, "Львів")]

# Завдання: Створити 'Student' namedtuple
Student = namedtuple('Student', ['first_name', 'last_name', 'age', 'city'])

# Рішення:
students = [Student(*data) for data in student_data]

# Тепер доступ чистий і читабельний
student_one = students[0]
print(f"{student_one.first_name} (вік {student_one.age}) живе в м.
{student_one.city}")
```

Завдання: `deque` (Черга FIFO)

Завдання: Ви симулюєте принтер, який отримує завдання на друк. Завдання мають виконуватися у тому порядку, в якому вони надійшли (FIFO). Використайте `deque` для реалізації цієї черги.

```
from collections import deque
import time

# Створюємо чергу завдань
print_queue = deque()

# Додаємо завдання (Enqueue)
print_queue.append("Документ1.pdf")
print_queue.append("Фото_відпустка.jpg")
print_queue.append("Презентація.pptx")
```

```
print(f"Завдання в черзі: {list(print_queue)}")

# Виконуємо завдання (Dequeue)
while print_queue:
    task = print_queue.popleft() # O(1) - це ключ!
    print(f"Друкується: {task}...")
    time.sleep(0.5)

print("Всі завдання надруковано.")
```

Завдання: `defaultdict` (Групування даних)

Завдання: У вас є список студентів та їхні оцінки. Потрібно швидко згрупувати студентів за їхньою оцінкою.

```
from collections import defaultdict

# Дано:
student_grades = [('Анна', 5), ('Петро', 4), ('Сергій', 5),
                  ('Марія', 3), ('Іван', 4)]

# Завдання: Згрупувати студентів за оцінкою
grouped_by_grade = defaultdict(list)

# Рішення:
for name, grade in student_grades:
    # Нам не потрібна перевірка "if grade not in grouped_by_grade"
    grouped_by_grade[grade].append(name)

# defaultdict(<class 'list'>, {5: ['Анна', 'Сергій'], 4: ['Петро', 'Іван'],
3: ['Марія']})
print(dict(grouped_by_grade))
print(f"Відмінники (5): {grouped_by_grade[5]}")
```

Завдання: `Counter` (Агрегація)

Завдання: Порахувати частоту кожного слова у реченні.

```
from collections import Counter

# Дано:
text = "паралельні обчислення це круто а функціональне програмування це
елегантно"

# Завдання: Порахувати частоту слів
words = text.split()
word_count = Counter(words)
```

```python
# Рішення:
print(word_count)

# Додаткова перевага:
print(f"2 найпоширеніші слова: {word_count.most_common(2)}")
```

Модуль 3: "Двигун" ФП в Python

Завдання: `Generators` (Ліниві обчислення)

Завдання: Уявіть, що у вас є лог-файл розміром 10 ГБ. Ви не можете завантажити його в пам'ять. Напишіть генератор для читання файлу рядок за рядком, який знаходить лише рядки з поміткою "ERROR".

```python
# (Уявімо, що "large_log.txt" - це наш великий файл)
def find_errors(log_file_path):
    print("--- Генератор почав роботу (файл відкрито) ---")
    with open(log_file_path, 'r', encoding='utf-8') as f:
        for line in f:
            if "ERROR" in line:
                # yield "заморожує" стан і повертає значення
                yield line.strip()
    print("--- Генератор завершив роботу (файл закрито) ---")

# Створюємо генератор (код ще не виконується)
error_lines = find_errors("large_log.txt")
# (для демонстрації створимо цей файл)
with open("large_log.txt", "w", encoding='utf-8') as f:
    f.write("INFO: Server started\n")
    f.write("DEBUG: User connected\n")
    f.write("ERROR: Failed to load module X\n")
    f.write("INFO: Processing request\n")
    f.write("ERROR: Database connection lost\n")

# Код виконується тільки тут, "на вимогу"
print("Шукаємо помилки:")
for error in error_lines:
    print(error)
```

Завдання: `Functions as First-Class Objects` (Диспетчер)

Завдання: Створити "калькулятор", який приймає назву операції у вигляді рядка та два числа. Використайте словник для зберігання самих функцій.

```python
def add(a, b):
    return a + b
```

```python
def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

# 1. Функції зберігаються у структурі даних (словнику)
operations_dispatcher = {
    'add': add,
    'subtract': subtract,
    'multiply': multiply
}

def calculate(operation_name, x, y):
    # 2. Отримуємо функцію зі словника і викликаємо її
    func = operations_dispatcher.get(operation_name)
    if func:
        return func(x, y)
    else:
        raise ValueError(f"Невідома операція: {operation_name}")

# Рішення:
print(f"Результат 'add': {calculate('add', 10, 5)}")
print(f"Результат 'multiply': {calculate('multiply', 10, 5)}")
```

Модуль 4: Функціональний інструментарій

Завдання: `lambda` (Сортування)

Завдання: У вас є список кортежів `(товар, ціна)`. Відсортуйте список за ціною, від найдешевшого до найдорожчого.

```python
# Дано:
products = [('Ноутбук', 32000), ('Мыша', 1200), ('Клавіатура', 2100)]

# Рішення:
# lambda x: x[1] - це анонімна функція,
# яка каже sorted() використовувати другий елемент (ціну) як ключ
sorted_products = sorted(products, key=lambda item: item[1])

print(sorted_products)
```

Завдання: `map` (Трансформація)

Завдання: У вас є список цін у гривнях. Створіть *новий* список, де всі ціни переведено у долари (умовно, $1 = 40 грн).

```python
# Дано:
prices_uah = [1200, 2100, 32000, 800]
UAH_TO_USD_RATE = 40

# Рішення:
prices_usd = list(map(lambda price: round(price / UAH_TO_USD_RATE, 2),
prices_uah))

print(f"Ціни в UAH: {prices_uah}")
print(f"Ціни в USD: {prices_usd}")
```

Завдання: `filter` (Фільтрація)

Завдання: У вас є список користувачів. Створіть *новий* список, який містить лише активних користувачів (`'status': 'active'`).

```python
# Дано:
users = [
    {'name': 'Анна', 'status': 'active'},
    {'name': 'Петро', 'status': 'inactive'},
    {'name': 'Сергій', 'status': 'active'},
    {'name': 'Марія', 'status': 'banned'}
]

# Рішення:
active_users = list(filter(lambda user: user['status'] == 'active', users))

print(active_users)
```

Завдання: `Comprehensions` ("Пітонічний" шлях)

Завдання: Зробіть те саме, що у `map` та `filter`, але за допомогою List Comprehensions.

1. Перевести ціни в UAH у USD.
2. Відфільтрувати активних користувачів.

```python
# 1. Заміна 'map'
prices_uah = [1200, 2100, 32000, 800]
UAH_TO_USD_RATE = 40
prices_usd_comp = [round(price / UAH_TO_USD_RATE, 2) for price in
prices_uah]
print(f"Ціни в USD (comprehension): {prices_usd_comp}")


# 2. Заміна 'filter' (і одразу 'map' для імен)
users = [
```

```
    {'name': 'Анна', 'status': 'active'},
    {'name': 'Петро', 'status': 'inactive'},
    {'name': 'Сергій', 'status': 'active'}
]
# Ми одночасно фільтруємо (if) і трансформуємо (user['name'])
active_user_names = [user['name'] for user in users if user['status'] ==
'active']
print(f"Імена активних (comprehension): {active_user_names}")
```

Завдання: `any` / `all` (Перевірки)

Завдання: Ви перевіряєте кошик покупця.

1. Чи `all` (всі) товари мають ціну > 0?
2. Чи `any` (хоча б один) товар має знижку?

```
# Дано:
cart = [
    {'name': 'Ноутбук', 'price': 32000, 'has_discount': False},
    {'name': 'Мыша', 'price': 1200, 'has_discount': True},
    {'name': 'Клавіатура', 'price': 2100, 'has_discount': False}
]

# 1. Чи всі ціни > 0?
all_prices_valid = all(item['price'] > 0 for item in cart)
print(f"Всі ціни валідні: {all_prices_valid}")

# 2. Чи є хоча б одна знижка?
any_discount = any(item['has_discount'] for item in cart)
print(f"Є знижки: {any_discount}")
```

Модуль 5: Просунуті патерни ФП

Завдання: `Closures` (Фабрика функцій)

Завдання: Створити функцію-"фабрику", яка генерує персоналізовані функції-привітання.

```
def greeting_maker(base_greeting):
    # 1. 'base_greeting' "заморожується" в замиканні

    def greeter(name):
        # 2. 'greeter' має доступ до 'base_greeting'
        return f"{base_greeting}, {name}!"

    # 3. Фабрика повертає нову функцію
    return greeter
```

```python
# Створюємо дві різні функції-привітання
greet_formal = greeting_maker("Доброго дня")
greet_informal = greeting_maker("Привіт")

# Використовуємо їх
print(greet_formal("Пане Володимире"))
print(greet_informal("Олено"))
```

Завдання: `Currying` (Часткове застосування)

Завдання: Створити гнучкий калькулятор податків. Базова функція `calculate_tax` має приймати `tax_rate` (ставку) і `amount` (суму). Перетворіть її на карровану функцію, щоб можна було "зафіксувати" ставку податку.

```python
# Каррована функція (Haskel Curry style)
def tax_calculator(tax_rate_percent):

    def apply_tax(amount):
        return amount * (tax_rate_percent / 100)

    return apply_tax

# Рішення: Створюємо спеціалізовані функції
calculate_pdv = tax_calculator(20) # 20% ПДВ
calculate_military_tax = tax_calculator(1.5) # 1.5% Військовий збір

# Використовуємо їх
salary = 50000
print(f"ПДВ з {salary}: {calculate_pdv(salary)}")
print(f"ВЗ з {salary}: {calculate_military_tax(salary)}")
```

Завдання: `Decorators` (Розширення функціонала)

Завдання: Написати декоратор `@timer`, який вимірює та виводить час виконання будь-якої функції.

```python
from functools import wraps
import time

def timer(func):
    @wraps(func) # <-- Важливо для збереження .__name__
    def wrapper(*args, **kwargs):
        start_time = time.perf_counter()

        # Виклик оригінальної функції
        result = func(*args, **kwargs)
```

```
        end_time = time.perf_counter()
        print(f"Функція '{func.__name__}' виконувалась {end_time -
start_time:.4f} сек.")
        return result
    return wrapper

# Рішення: Застосовуємо декоратор
@timer
def process_data(n):
    """Дуже важлива функція, що довго працює."""
    sum = 0
    for i in range(n):
        sum += i
    return sum

# Викликаємо функцію як зазвичай
result = process_data(1_000_000)
print(f"Ім'я функції: {process_data.__name__}") # Завдяки @wraps ім'я
коректне
print(f"Докстрінг: {process_data.__doc__}")
```

Модуль 6: Додаток — `decimal`

Завдання: `decimal` (Точні розрахунки)

Завдання: Ви зіткнулися з класичною помилкою `0.1 + 0.2`. Потім вам потрібно порахувати суму кошика, де ціни задані як рядки. Використайте `Decimal` для точності.

```
from decimal import Decimal, getcontext

# 1. Проблема з float
print(f"Проблема: 0.1 + 0.2 = {0.1 + 0.2}")

# 2. Рішення з Decimal
# Створюємо з рядків для абсолютної точності
a = Decimal("0.1")
b = Decimal("0.2")
print(f"Рішення: {a} + {b} = {a + b}")
print("-" * 20)

# 3. Практичне завдання: підрахунок суми кошика
# (Напр., ціни прийшли з JSON у вигляді рядків)
cart_prices = ["199.99", "1.50", "10.00", "0.33"]

total_cost = Decimal("0.00")
for price in cart_prices:
    total_cost += Decimal(price)
```

```python
print(f"Точна сума кошика: {total_cost} UAH")

# Округлення для користувача (банківське округлення)
getcontext().prec = 4 # Встановимо точність у 4 знаки
print(f"Округлена сума: {total_cost.quantize(Decimal('0.01'))}")
```