Principles of Software Engineering and Data Bases

Davide Yi Xian Hu
Email: davideyi.hu@polimi.it
Date: 11 December 2024
Exercise Lecture: 09 - Testing

POLITECNICO
MILANO 1863

Functional Testing

Exercise 1 - Is Palindrome

Specification

Function Name: is_palindrome
Input: A string (s).
Output: Boolean (True if s is a palindrome, False otherwise).
Definition: A string is a palindrome if it reads the same backward as forward, ignoring case and spaces.

Exercise 2 - Find Max

Specification

Function Name: find_max
Input: A list of integers (nums).
Output: The maximum integer in the list.
Behavior: Return the maximum element in the list.
If the list is empty, raise a ValueError.

Exercise 3 - Factorial

Specification

Function Name: factorial
Input: A non-negative integer (n).
Output: An integer representing the factorial of n.
Behavior: The factorial of n is defined as:

- $n! = n \times (n-1) \times (n-2) \times \ldots \times 1 =$ (for $n > 0$).
- $0! = 1$.

If n is negative, raise a ValueError.

Exercise 4 - Check Prime Number

Specification

Function Name: is_prime
Input: A value (n).
Output: Boolean (True if n is a prime number, False otherwise).
Behavior: A prime number is greater than 1 and divisible only by 1 and itself.
The function should return False for any non-positive numbers (e.g., 0, negative numbers).
If the input is not an integer, raise a TypeError.

Structural Testing

Exercise 5 - Discounted Price

```
def find_discounted_price(price, discount_code):
    """Calculate the discounted price based on the discount code."""
    if price ⩽ 0:
        return "Invalid price"

    if discount_code == "DISCOUNT10":
        price *= 0.9  # Apply 10% discount
    elif discount_code == "DISCOUNT20":
        price *= 0.8  # Apply 20% discount
    elif discount_code == "DISCOUNT50":
        price *= 0.5  # Apply 50% discount
    else:
        return "Invalid discount code"

    if price < 1:
        return "Discount too large"

    return round(price, 2)
```

Exercise 6 - Is Leap Year

```
def is leap year(year):
    """Determine if a given year is a leap year."""
    if not isinstance(year, int) or year <= 0:
        raise ValueError("Year must be a positive integer")

    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Exercise 7 - Is Palindrome (again...)

```
def is_palindromic_number(n):
    """Check if the given number is a palindromic number."""
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")

    if n < 0:
        return False

    str_n = str(n)
    return str_n == str_n[::-1]
```

Exercise 8 - Loan Repayment Plan

```
def calculate_loan_plan(principal, rate, months, plan_type):
    """Calculate loan repayment details based on the given plan."""
    if not isinstance(principal, (int, float)) or principal <= 0:
        raise ValueError("Principal must be a positive number")
    if not isinstance(rate, (int, float)) or rate < 0:
        raise ValueError("Rate must be a non-negative number")
    if not isinstance(months, int) or months <= 0:
        raise ValueError("Months must be a positive integer")
    if plan_type not in ["standard", "accelerated", "interest-only"]:
        raise ValueError("Invalid plan type")

    monthly_rate = rate / 100 / 12
    total_repayment = 0
    total_interest = 0
    monthly_payment = 0
    balance = principal

    if plan_type == "standard":
        # Calculate fixed monthly payment using the formula:
        # M = P * r * (1 + r)^n / ((1 + r)^n - 1)
        if monthly_rate == 0:
            monthly_payment = principal / months
        else:
            monthly_payment = (
                principal
                * monthly_rate
                * (1 + monthly_rate) ** months
                / ((1 + monthly_rate) ** months - 1)
            )
        total_repayment = monthly_payment * months
        total_interest = total_repayment - principal
"""
    elif plan_type == "accelerated":
        # Payments increase by 1% each month
        month = 1
        while month <= months:
            monthly_payment = (principal / months) * (1 + 0.01 *
month)
            interest = balance * monthly_rate
            total_interest += interest
            balance -= (monthly_payment - interest)
            total_repayment += monthly_payment
            month += 1

    elif plan_type == "interest-only":
        # Pay only the interest each month, principal at the end
        month = 1
        while month < months:
            monthly_payment = principal * monthly_rate
            total_interest += monthly_payment
            total_repayment += monthly_payment
            month += 1
        total_repayment += principal  # Add the principal at the
end

    return {
        "monthly_payment": round(monthly_payment, 2),
        "total_repayment": round(total_repayment, 2),
        "total_interest": round(total_interest, 2),
    }
```

Regression Testing

Exercise 9 - E-commerce Application

```
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, item_name, price, quantity=1):
        if item_name in self.items:
            self.items[item_name]['quantity'] += quantity
        else:
            self.items[item_name] = {'price': price, 'quantity': quantity}

    def remove_item(self, item_name):
        if item_name in self.items:
            del self.items[item_name]

    def get_total_price(self):
        return sum(item['price'] * item['quantity'] for item in self.items.values())

    def get_items(self):
        return self.items
```

Exercise 9 - E-commerce Application (new version)

```python
class ShoppingCart:
    def __init__(self):
        self.items = {}
        self.discount = 0  # New attribute for discounts

    def add_item(self, item_name, price, quantity=1):
        if item_name in self.items:
            self.items[item_name]['quantity'] += quantity
        else:
            self.items[item_name] = {'price': price, 'quantity': quantity}

    def remove_item(self, item_name):
        if item_name in self.items:
            del self.items[item_name]

    def set_discount(self, percentage):
        """Set a discount as a percentage."""
        if not (0 ⩽ percentage ⩽ 100):
            raise ValueError("Discount must be between 0 and 100")
        self.discount = percentage

    def get_total_price(self):
        total = sum(item['price'] * item['quantity'] for item in self.items.values())
        return total * (1 - self.discount / 100)

    def get_items(self):
        return self.items
```

Exercise 10 - Bank Account

```
class BankAccount:
    def __init__(self, initial_balance=0):
        if initial_balance < 0:
            raise ValueError("Initial balance cannot be negative")
        self.balance = initial_balance

    def deposit(self, amount):
        if amount ⩽ 0:
            raise ValueError("Deposit amount must be positive")
        self.balance += amount

    def withdraw(self, amount):
        if amount ⩽ 0:
            raise ValueError("Withdrawal amount must be positive")
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount

    def get_balance(self):
        return self.balance
```

Exercise 10 - Bank Account (new version)

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        if initial_balance < 0:
            raise ValueError("Initial balance cannot be negative")
        self.balance = initial_balance
        self.transaction_fee = 2  # New feature: Transaction fee

    def deposit(self, amount):
        if amount ⩽ 0:
            raise ValueError("Deposit amount must be positive")
        self.balance += amount

    def withdraw(self, amount):
        if amount ⩽ 0:
            raise ValueError("Withdrawal amount must be positive")
        total_amount = amount + self.transaction_fee
        if total_amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= total_amount

    def get_balance(self):
        return self.balance
```