BABEŞ-BOLYAI UNIVERSITY
Faculty of Mathematics and Computer Science

Algorithms and Programming

Lecture 12 – Recap

Camelia Chira

Course content

Programming in the large

1. Software development process
2. Procedural programming
3. Modular programming
4. Abstract data types
5. Software development principles
6. Testing and debugging

Programming in the small

7. Recursion
8. Complexity of algorithms
9. Search and sorting algorithms
10. Problem solving methods

1 Software development process

Programming in the large

- Programming
  - Python programs
  - Data types
  - List operations
  - Assignments
  - Statements
  - Control flow
- Software development process
- Feature-driven development

2

Procedural programming

Programming in the large

- Programming paradigms
- Procedural programming
- Function definition
- Variable scope
- Test-driven development

3

Modular programming

Programming in the large

- Modules
- Packages
- import statement

- How to organize an application
- Layered architecture
- Exceptions
  - raise statement
  - try..except(..finally) statement

3 Modular programming

Programming in the large

- Module
  - Structural unit (that can communicate with other units), changeable
  - Collections of functions and variables that implement a well-defined feature
- Based on decomposing the problem in subproblems considering:
  - Separating concepts
  - Layered architectures
  - Maintenance and reuse of code
  - Cohesion of elements in a module
  - Link between modules

4

Abstract Data Types

Programming in the large

- Abstract Data Type
- Classes
- Data abstraction
- Encapsulation
- Information hiding
- Class attributes vs. instance attributes
- Static methods
- UML diagrams

5

Software development principles

Programming in the large

- Design principles
- Layered architecture

- Key design principles
  - Single Responsibility Principle
  - Separation of Concerns Principle
  - Reuse Principle
  - Cohesion and Coupling Principle

GRASP

- General Responsibility Assignment Software Patterns (GRASP)
  - Guidelines for assigning responsibility to classes and objects

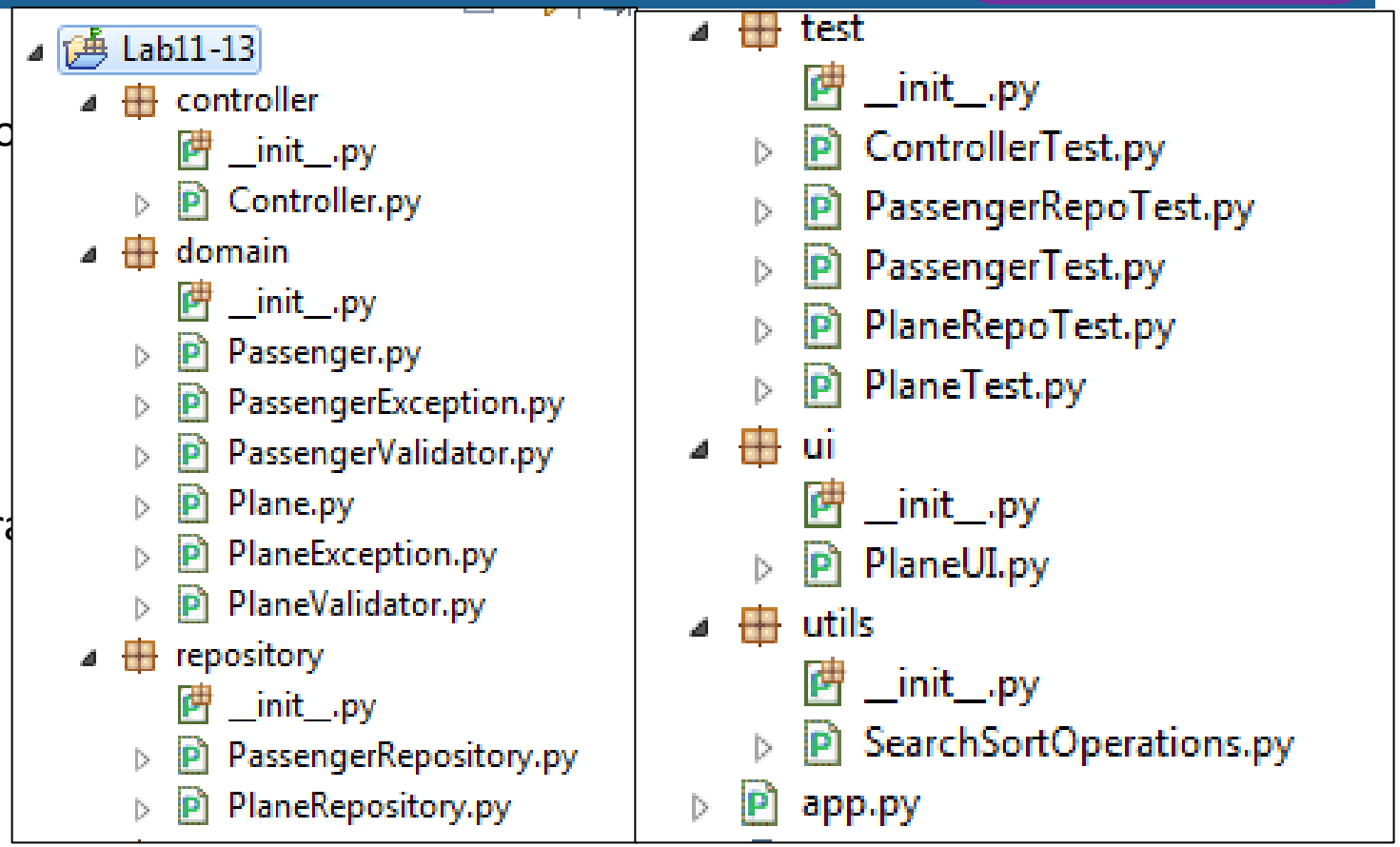
- High Cohesion
- Low Coupling
- Information Expert
- Creator
- Pure Fabrication
- Controller

Layered architecture:

- **High cohesion**
  To increase cohesion: break programs into classes and subsystems
  Low cohesion means that an element has too many unrelated responsibilities => problems: hard to understand, hard to reuse, hard to maintain
- **Low coupling**
  Low dependency between classes
  Low impact in a class of changes in other classes
  High reuse potential

Layered architecture

- User Interface
  - Functions, modules, classes fo
  - *UI / View / Presentation*
- Domain
  - The logic of the application
  - *Business Logic / Model*
- Infrastructure
  - Functions with a general chara
  - *Utils*
- Coordinator
- Controller
- Repository
- Test



6

Testing and debugging

Programming in the large

- Blackbox testing
- Whitebox testing

- Testing levels
- Testing in Python: unittest

7

Recursion

Programming in the small

- A programming technique where a function calls itself
- Basic concepts
  - Recursive element – an element that is defined by itself
  - Recursive algorithm – an algorithm that calls itself
- Recursion can be:
  - **Direct** – a function calls itself (`f` calls `f`)
  - **Indirect** – a function `f` calls a function `g`, function `g` calls `f`
- Main idea of developing a recursive algorithm for a problem of size n
  - **Base case**
    - How to stop recursion
    - Identify the base case solution (for n=1)
  - **Inductive step**
    - Break the problem into a simpler version of the same problem plus some other steps

8 Complexity of algorithms

Programming in the small

- Complexity in time
  - Running time of an algorithm:
    - It is not a fixed number but a function *T(n)* that depends on the size *n* of the input data
    - Measures the basic steps the algorithm makes
  - Best case, Worst case, Average case
  - Exact steps vs Big Oh or O() notation
  - O(n) measure
    - How the running time grows depending on the input data size
    - Expression for the number of operations -> asymptotic behavior as the problem gets bigger
- Complexity in space
  - Estimates the **space** (memory) that an algorithm needs to store input data, output data and any temporary data

Search methods

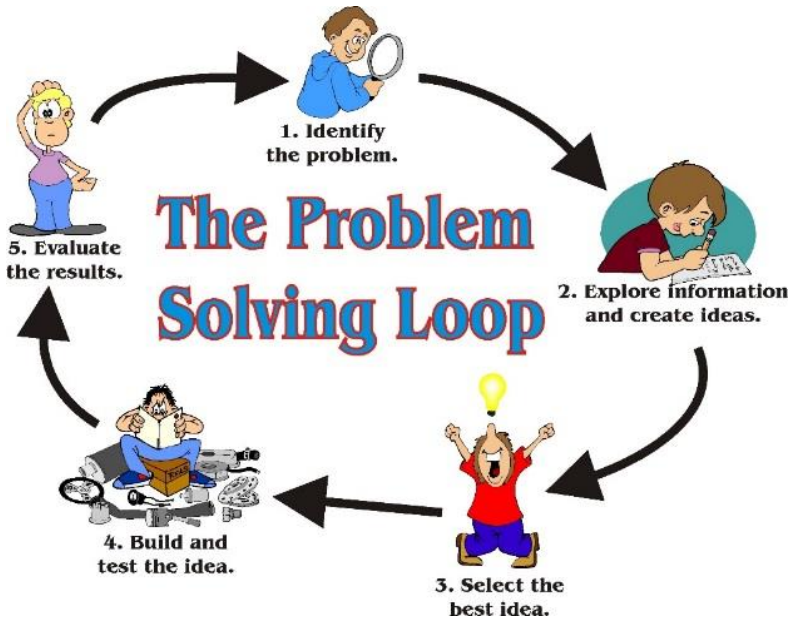
- **Sequential search**
  - Basic idea: the elements of the list are examined one by one (the list can be ordered or not)
  - O(n)
- **Binary search**
  - Basic idea: the problem is divided in two similar but smaller subproblems (the list has to be ordered)
  - O(log n)
- Python
  - Functions index, count, find

Sorting methods

- Selection sort
  - Swap the smallest element with the first one & repeat for all elements
- Insertion sort
  - Insert each element at the correct position in a sublist with the elements already sorted
- Bubble sort
  - Compare any 2 consecutive elements and swap them if not in correct order
- Quick sort
  - Divide and conquer: divide the list in 2 parts and sort the sublists
- Python functions
  - sort, sorted

Problem solving methods

- Solving problems by search using standard methods
  - Exact methods
    - **Generate and test**
    - **Backtracking**
    - **Divide and conquer**
    - **Dynamic programming**
  - Heuristic methods
    - **Greedy method**



Reading materials and useful links

1. The Python Programming Language - https://www.python.org/
2. The Python Standard Library - https://docs.python.org/3/library/index.html
3. The Python Tutorial - https://docs.python.org/3/tutorial/
4. M. Frentiu, H.F. Pop, Fundamentals of Programming, Cluj University Press, 2006.
5. MIT OpenCourseWare, Introduction to Computer Science and Programming in Python, https://ocw.mit.edu, 2016.
6. K. Beck, Test Driven Development: By Example. Addison-Wesley Longman, 2002. http://en.wikipedia.org/wiki/Test-driven_development
7. M. Fowler, Refactoring. Improving the Design of Existing Code, Addison-Wesley, 1999. http://refactoring.com/catalog/index.html

Exam

- **Written exam (40% of final grade)**
  - Group **811**: **Friday 20.01.2023**, room **6/II** from **12:00**
  - Group **812**: **Friday 20.01.2023**, room **6/II** from **13:00**
  - Group **813**: **Friday 20.01.2023**, room **2/I** from **14:00**

  To attend, you need to have the minimum required attendance for seminar (75%) and lab (90%)

- **Practical exam (30% of final grade)**

Date and time of your last lab (with your own semigroup)

Retake Exam

The **retake exam** will take place as follows:

- **Written** exam: **Tuesday 21.02.2023**, room **6/II** from **10:00** for all groups
- **Practical** exam: **Tuesday 21.02.2023**, room **L308** from **14:00** for all groups

Written exam

- Time: 50 minutes
- **Types of questions**
  - Open ended and/or multiple choice questions
- Both theoretical and practical questions

Sample Exam Questions

- **Examples of theoretical questions**
  - *What is test-driven development?*
  - *What is the difference between a local variable and a global variable?*
  - *Define and explain the backtracking method.*

Sample Exam Questions

- **Examples of practical questions**

Write the code to select the elements of a list (containing string values) for which the first 3 letters are 'The' and return them as a new list. Use the Python function filter and consider that the name of the given list is 'lst'. (5 Points)

5

What is printed on the screen when executing the code given in the image? (5 Points)

```
a = 1
def f():
    a = 2
    print("a = ", a)
f()
print("a = ", a)
```

Sample Exam Questions

Consider the function f defined as given in the image. Choose all statements that are correct.
(5 Points)

```
def f(s):
    return s[0:] == s[::-1]
```

- [ ] f("ana") returns True
- [ ] f("Hannah") returns True
- [ ] f([1]) returns True
- [ ] f("") returns False
- [ ] f("Ana") returns False

Sample Exam Questions

```
def f(l):
    if len(l) == 0:
        return []
    else:
        return [l[-1]] + f(l[:-1])
```

What is the function f doing?
OR
Write the documentation of f.
OR
Test the function f.

```
def f(n):
    if n == 0:
        return []
    else:
        ...
```

Consider the code given in the image. The function f should be a recursive function that returns a list with the digits of the number n given as argument, in reverse order. For instance, for n=123 the result should be [3, 2, 1]. Write the code that is missing from the image (a single line of code).

Sample Exam Questions

```
lst = [1, 2]
try:
    lst.append(3)
    n = len(lst)
    for i in range(n):
        lst[i] = lst[i] / (n-i-1)
    lst.append(4)
except:
    print("Exception")
print(lst)
```

What is printed on the screen after executing the code in the given image?
Explain your answer.

```
class Movie():
    def __init__(self, title, year):
        self.__title = title
        self.__year = year
```

Information hiding ? Data abstraction ?

Create a repository of Movie objects and sort it by title.

Filter the repository to get all movies from 2022.

Sort movies from 2022 by title.

Q & A