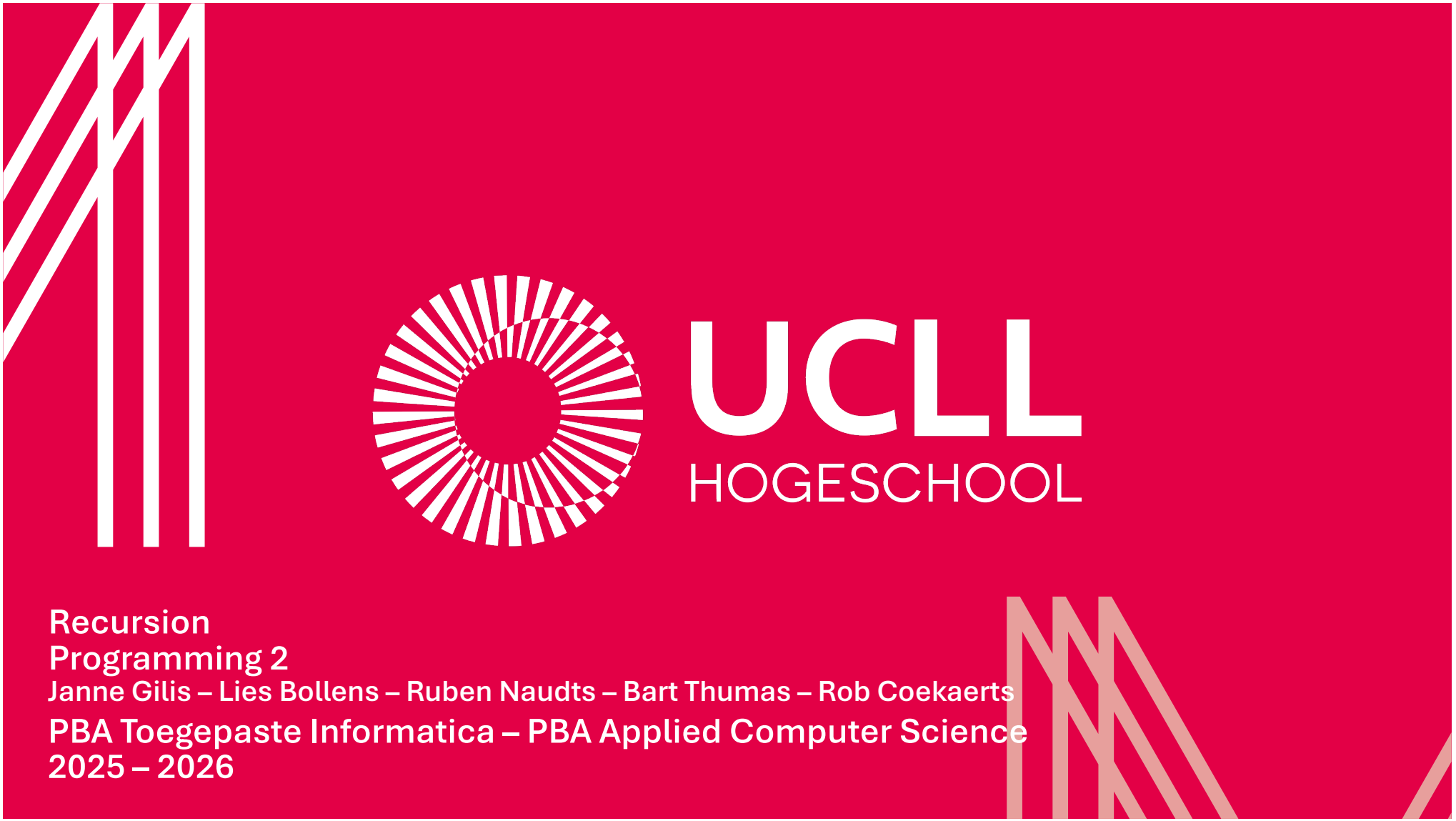


Recursion
Programming 2

Janne Gilis – Lies Bollens – Ruben Naudts – Bart Thumas – Rob Coekaerts

PBA Toegepaste Informatica – PBA Applied Computer Science
2025 – 2026

Agenda

What is recursion in programming

Iteration VS Recursion

Construction of recursive algorithm

Example Factorial

When do we use recursion?

Introduction to recursion

Before diving into the definition, let's start with a simple example. Consider calculating the sum of the first 'n' natural numbers recursively.

```
def sum_n(n):
    if n == 1:
        return 1
    else:
        return n + sum_n(n - 1)

print(sum_n(5))  # Output: 15
```

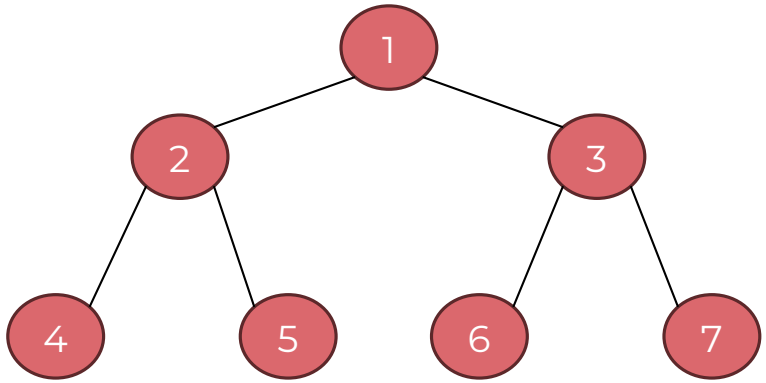
What is recursion in programming

- Recursion is a technique where a function calls to itself within its definition. This process continues until a base case is reached, which is a condition where the function returns a value without making any further recursive calls. Without a base case, the recursion would continue indefinitely, leading to what's known as "infinite recursion," which can cause the program to crash.

1. Base Case – The condition that stops the recursion.
2. Recursive Case – The function calls itself with modified arguments to approach the base case.

Example Depth first search

- Depth-First Search (DFS) is a traversal algorithm that explores as far as possible along each branch before backtracking.




```python
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def dfs(node):
    """Simple recursive depth-first search (preorder traversal)"""
    if not node:
        return  # Base case: stop when reaching a None node

    print(node.value)  # Visit the current node
    dfs(node.left)  # Recursively visit left subtree
    dfs(node.right)  # Recursively visit right subtree

# Example: Creating a simple binary tree
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

# Run the recursive DFS function
print("Depth-First Search (DFS) Recursive:")
dfs(root)
```

Iteration vs recursion

- Print all integers from the given number a to the given number b.
- Iteration

```
def function(a, b):
    for i in range(a, b + 1):
        print(i)
```

- Recursion

```
def function(a, b):
    if a > b:
        return
    print(a)
    function(a + 1, b)
```

Iteration vs recursion

Feature	Iteration	Recursion
Definition	Repetition using loops	Function calls itself
Use Cases	Simpler problems	Complex divide-and-conquer
Performance	Lower memory usage	Higher memory usage
Requires Stack?	No	Yes

Construction of recursive algorithm

- Base case: writing as an exception rule
  - When are we done?
  - What is the simplest case we can think of?
  - What is the answer for this simplest case?

- Recursive case
  - What is added/taken away to arrive at the base case?

Example factorial

Function will call itself and repeat its behaviour until some condition is met to return a result

```
def factorial_recursive(n):

 # Base case:
if n == 1:
  return 1

# Recursive case:
else:
  return n * factorial_recursive(n-1)
```

Example Factorial

- factorial_recursive(5)
- 5 * factorial_recursive(4)
- 5 * 4 * factorial_recursive(3)
- 5 * 4 * 3 * factorial_recursive(2)
- 5 * 4 * 3 * 2 * factorial_recursive(1)
- 5 * 4 * 3 * 2 * 1
- = 120

Recursive vs. Iterative Approach

- Recursion and iteration both achieve repetition but work differently.
  - Recursive Approach: A function calls itself until a base case is met
  - Iterative Approach: Uses loops (for, while) to repeat operations.

Recursive:

```python
def factorial_recursive(n):
    if n == 1:
        return 1
    return n * factorial_recursive(n - 1)
```

Iterative:

```python
def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

When do we use recursion?

- **The problem has a natural recursive structure.**
  - If solving the problem involves solving smaller versions of the same problem, recursion is a good choice.
- **You are working with recursive data structures.**
  - Data structures like trees, graphs, and linked lists often require recursion for efficient traversal.
- **Divide-and-Conquer algorithms.**
  - If a problem can be divided into independent subproblems, recursion helps break it down. Examples:
    - Merge Sort
    - Quick Sort
    - Binary Search

Example

- When Recursion is Useful (Tree Traversal)

```python
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def dfs(node):
    """Simple recursive depth-first search (preorder traversal)"""
    if not node:
        return  # Base case: stop when reaching a None node

    print(node.value)  # Visit the current node
    dfs(node.left)  # Recursively visit left subtree
    dfs(node.right)  # Recursively visit right subtree
```

This is natural recursion, as trees are recursive structures.

Example

- When Iteration is Better (Sum of First N Numbers)

Using Recursion:

```
def sum_n(n):
    if n == 0:
        return 0
    return n + sum_n(n - 1)
```

Using Iteration (Better)

```
def sum_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
```

- The iterative version is better because it avoids unnecessary function calls and memory usage.

Exercise

Try the following exercises

- 02-functional-programming
  - 07-recursion

