ILP 2023 – Extra
About Recursion

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Recursion: definition

- **Definition (recursion):** Recursion is a common mathematical and programming concept. It means that a **function can be defined by calling itself**.

Recursion: definition

- **Definition (recursion):** Recursion is a common mathematical and programming concept. It means that a **function can be defined by calling itself**.

- An example of a recursive function is the **factorial function** with value n (denoted $\boldsymbol{n}!$).

$$n! = 1 \times 2 \times \cdots \times n$$

Recursion: definition

- **Definition (recursion):** Recursion is a common mathematical and programming concept. It means that a **function can be defined by calling itself**.

- An example of a recursive function is the **factorial function** with value n (denoted $\boldsymbol{n}!$). You typically know it as

$$n! = 1 \times 2 \times \cdots \times n$$

- The factorial function value can be defined in a recursive manner, as follows.

$$n! = \begin{cases} 1 & if\ n = 1 \\ n \times ((n-1)!) & if\ n > 1 \end{cases}$$

Recursion: definition

- **Definition (recursion):** Recursion is a common mathematical and programming concept. It means that a **function can be defined by calling itself**.

- An example of a recursive function is the **factorial function** with value n (denoted $\boldsymbol{n}!$). You typically know it as

$$n! = 1 \times 2 \times \cdots \times n$$

- The factorial function value can be defined in a recursive manner, as follows.

$$n! = \begin{cases} 1 & if\ n = 1 \\ n \times \left((n-1)!\right) & if\ n > 1 \end{cases}$$

- In other words, we can define the value of the factorial function with value n, by reusing the factorial function with value $(n-1)$.

Recursion vs. for loop

- The **factorial function** with value n (denoted $\boldsymbol{n}!$) is defined as
$$n! = 1 \times 2 \times \cdots \times n$$

- Following this definition, we could compute the value of the factorial using a simple for loop.

- And that would work just fine.

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

```
print(factorial_fun_for(1))
```

```
1
```

```
print(factorial_fun_for(3))
```

```
6
```

```
print(factorial_fun_for(5))
```

```
120
```

Recursion vs. for loop

- But we could also use the recursive definition for the factorial function, defined as

$$n! = \begin{cases} 1 & if\ n = 1 \\ n \times \left((n-1)!\right) & if\ n > 1 \end{cases}$$

- And it works as well, without any **for** loop!

```
def factorial_fun_rec(n):
    if n == 1:
        value = 1
    else:
        value = n*factorial_fun_rec(n - 1)
    return value
```

```
print(factorial_fun_rec(1))
```

1

```
print(factorial_fun_rec(3))
```

6

```
print(factorial_fun_rec(5))
```

120

Recursion vs. for loop

- What happens in practice? Multiple **concurrent runs** of a function.

```python
def factorial_fun_rec(n):
    print("Function called, with n = {}".format(n))
    if n == 1:
        value = 1
    else:
        value = n*factorial_fun_rec(n - 1)
    print("Function completed, with n = {}, and return value = {}".format(n, value))
    return value
```

```python
print(factorial_fun_rec(1))
```

```
Function called, with n = 1
Function completed, with n = 1, and return value = 1
1
```

```python
print(factorial_fun_rec(3))
```

```
Function called, with n = 3
Function called, with n = 2
Function called, with n = 1
Function completed, with n = 1, and return value = 1
Function completed, with n = 2, and return value = 2
Function completed, with n = 3, and return value = 6
6
```

Recursion vs. for loop

- **Recursion** is an interesting trick for computing the value of a function, which would normally require an **iterative for loop**.

Recursion vs. for loop

- **Recursion** is an interesting trick for computing the value of a function, which would normally require an **iterative for loop**.

- In some cases, the value of a function **can only be computed** using a recursive approach.

Recursion vs. for loop

- **Recursion** is an interesting trick for computing the value of a function, which would normally require an **iterative for loop**.

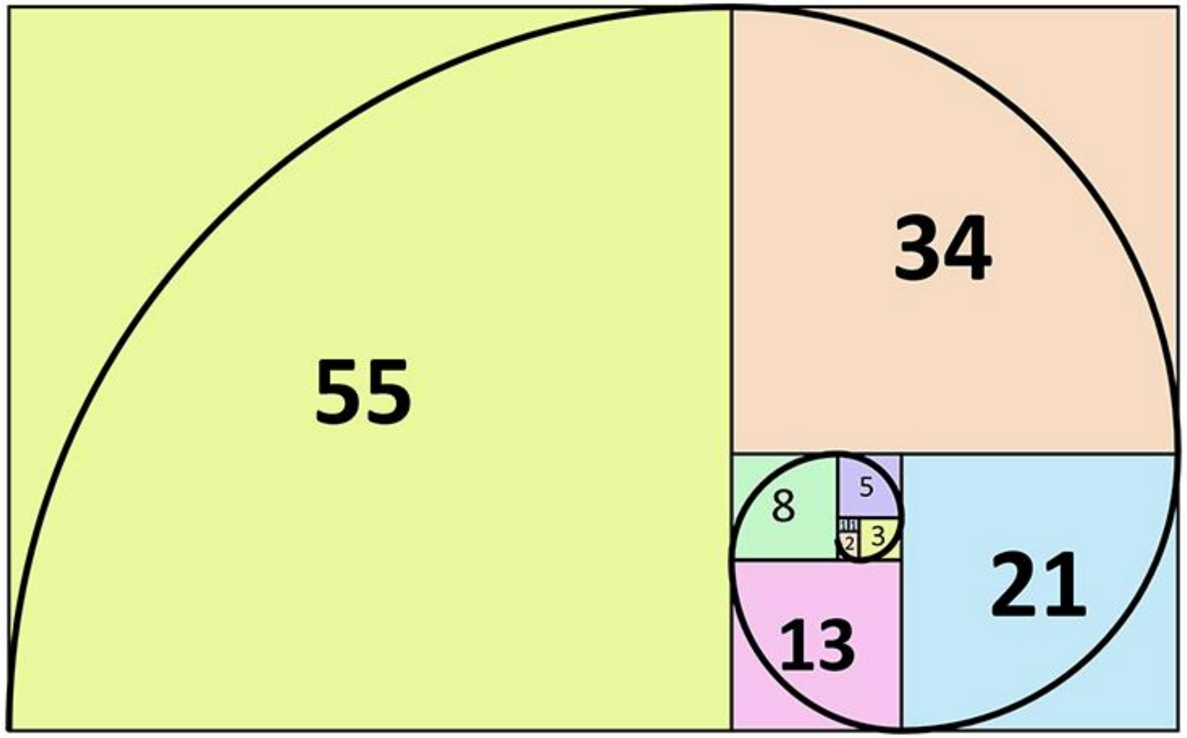
- In some cases, the value of a function **can only be computed** using a recursive approach.
- In some cases, recursion might be **faster** than an iterative for loop. Sometimes it is not.

→ Something we will discuss once we investigate computational complexity (a.k.a. the science of designing the best code for a task)

Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

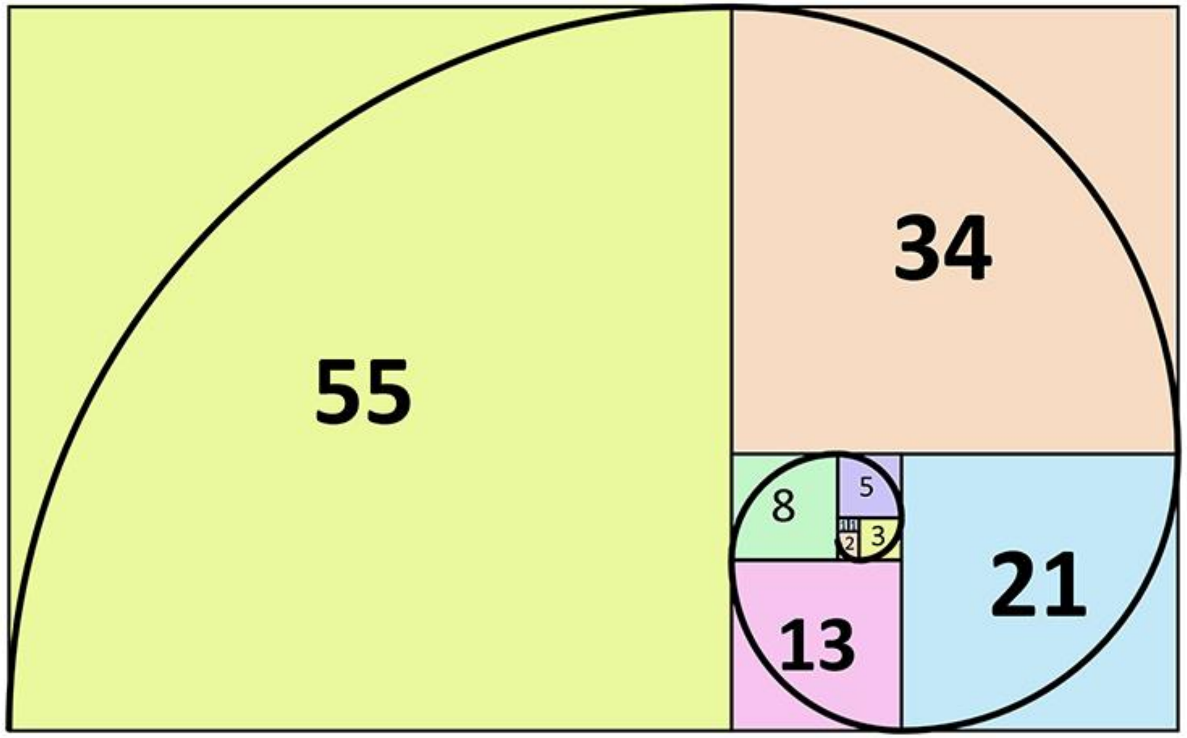


Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

The Fibonacci sequence F is a mathematical curiosity.



Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

The Fibonacci sequence F is a mathematical curiosity.

- The first and second elements of the sequence are equal to 1, i.e. $F(1) = 1$ and $F(2) = 1$.

Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

The Fibonacci sequence F is a mathematical curiosity.

- The first and second elements of the sequence are equal to 1, i.e. $F(1) = 1$ and $F(2) = 1$.
- The n-th Fibonacci element, with $n \geq 3$, is defined as the **sum of its previous two elements**, i.e. $F(n) = F(n-1) + F(n-2)$.

Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

The Fibonacci sequence F is a mathematical curiosity.

- The first and second elements of the sequence are equal to 1, i.e. $F(1) = 1$ and $F(2) = 1$.

- The n-th Fibonacci element, with $n \geq 3$, is defined as the **sum of its previous two elements**, i.e. $F(n) = F(n-1) + F(n-2)$.

- **Task:** Write **two functions** that compute and return the n-th Fibonacci element, for any value of $n \geq 1$.

Practice activities: **recursion** vs. for loops

Let us practice a bit

Activity 5 – Fibonacci sequence.ipynb

The Fibonacci sequence F is a mathematical curiosity.

- The first and second elements of the sequence are equal to 1, i.e. $F(1) = 1$ and $F(2) = 1$.

- The n-th Fibonacci element, with $n \geq 3$, is defined as the **sum of its previous two elements**, i.e. $F(n) = F(n-1) + F(n-2)$.

- **Task:** Write **two functions** that compute and return the n-th Fibonacci element, for any value of $n \geq 1$.

→ The first function uses a **for** loop, the second uses **recursion**.