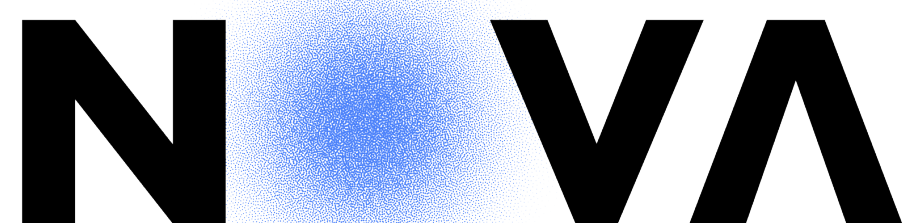


Linguagens e Ambientes de Programação

LEI - Licenciatura em Engenharia Informática

João Costa Seco (joao.seco@fct.unl.pt)

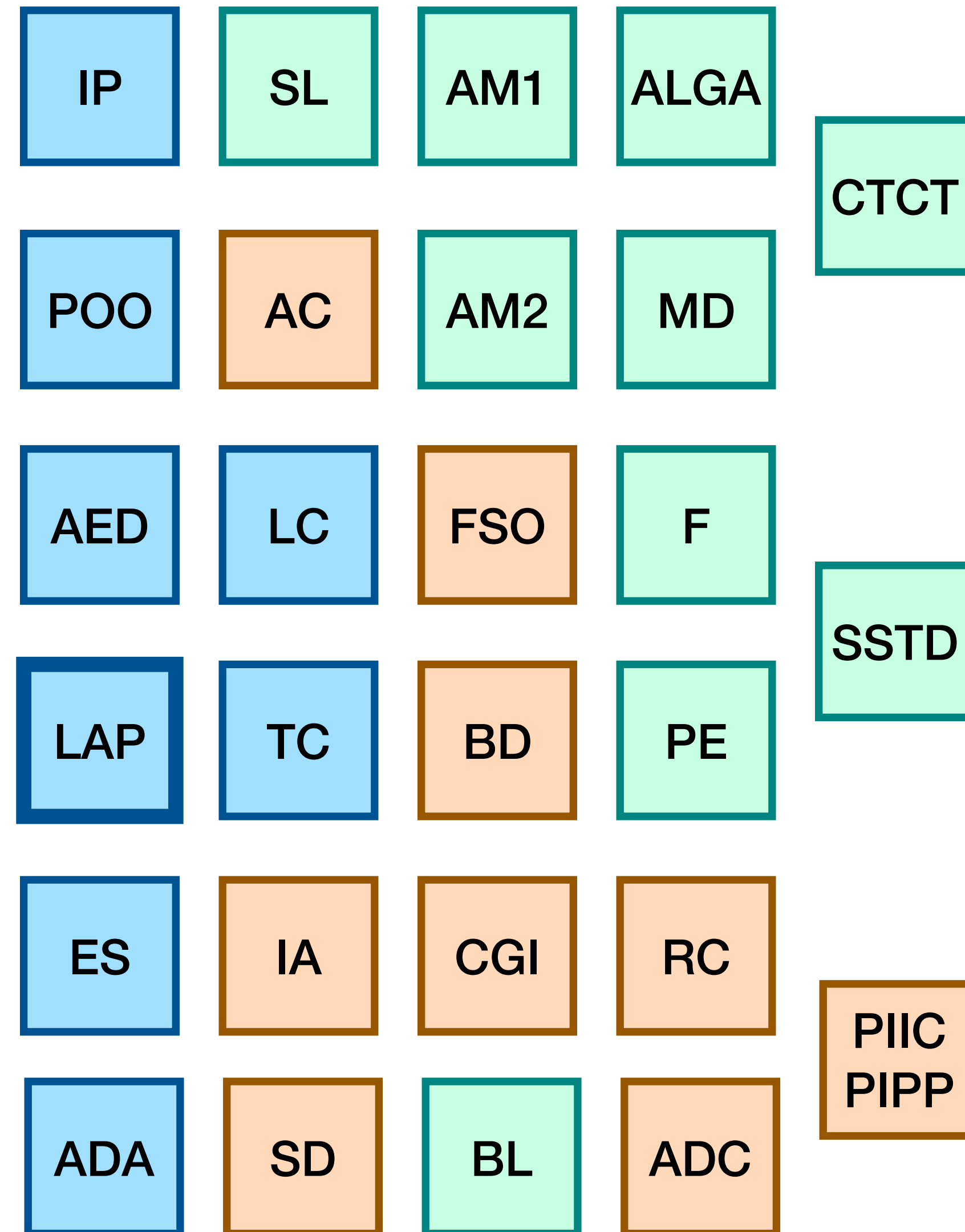


NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

LAP 2024

A Unidade Curricular

LAP na LEI



Linguagens e Ambientes de Programação 2024

1. Estudo de um novo paradigma: programação funcional
2. Estudo dos fundamentos das linguagens de programação
3. A programação funcional é o veículo de comunicação para as LP
4. Aprender Programação Funcional com OCaml
5. Aprender muitas linguagens de programação numa só:
 - fazendo, e comparando.

Porquê estudar linguagens de programação?

Why (1): As linguagens de programação são a “nossa ferramenta”.

Why (2): Cada linguagem é apropriada a um estilo de programas e sistemas. É preciso conhecer bem, para escolher melhor.

Why (3): As linguagens de programação modernas usam/misturam muitos conceitos diferentes (paradigmas).

Why (4): As linguagens também são modelos precisos de raciocínio (programas que simulam sistemas provam que eles são possíveis).

Why (5): Desenhar novas linguagens é mais comum do que se pensa.

Why (6): Quem sabe o porquê das coisas, faz melhor.

Porquê estudar programação funcional (tipificada)?

Why (1): as LP funcionais têm uma semântica precisa

Why (2): os sistemas de tipos permitem raciocínios avançados

Why (3): são linguagens sem interferências e casos especiais

(e.g. visibilidade de nomes, aliasing, efeitos laterais, conversões implícitas).

Why (4): ideais para compreender e desenhar outras linguagens

Why (5): imutabilidade de estado (é possível raciocinar sem ser por debug).

Why (6): modularidade e composicionalidade

Why (7): abstração de ordem superior (código genérico).

Why (8): segurança de tipos (não há nulls, não há (tantos) erros de execução).

Conceitos de linguagens

- Sintaxe, semântica e pragmática
- Linguagens declarativas vs linguagens imperativas
- Declaração e definição de nomes (ligação, ambiente, âmbito)
- Polimorfismo paramétrico e universal
- Abstração e parametrização / Ligação (closures)
- Modularidade e composição
- Tipos algébricos (e indutivos)
- Sistemas de tipos (soundness), inferência de tipos (linguagens com ...)
- Raciocínio e correção (testes vs verificação)
- Modelo de execução (stack based) - tail recursive
- Lazy vs Strict
- Estruturas de dados persistentes (why?)
- Estado, aliasing, etc. (pitfalls)
- Interpretadores e Compiladores (Just-in-time)

Programação Funcional em Java

```
Collections.sort(numbers, new Comparator<Integer>() {  
    @Override  
    public int compare(Integer n1, Integer n2) {  
        return n1.compareTo(n2);  
    }  
});
```

```
Collections.sort(numbers, (n1, n2) -> n1.compareTo(n2));
```


Programação Funcional em C

```
#include <stdio.h>
#include <stdlib.h>

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b) {
    return ( *(int*)a - *(int*)b );
}

int main () {
    int n;

    printf("Before sorting the list is: \n");
    for( n = 0 ; n < 5; n++ ) {
        printf("%d ", values[n]);
    }

    qsort(values, 5, sizeof(int), cmpfunc);

    printf("\nAfter sorting the list is: \n");
    for( n = 0 ; n < 5; n++ ) {
        printf("%d ", values[n]);
    }

    return(0);
}
```

Programação Funcional em Scala

```
case class ChessPiece(color: Color, name: Name)
val rook = ChessPiece(White, Rook)
```

```
sealed trait Color
final case object White extends Color
final case object Black extends Color
```

```
sealed trait Name
final case object Pawn extends Name
final case object Rook extends Name
final case object Knight extends Name
final case object Bishop extends Name
final case object Queen extends Name
final case object King extends Name
```

```
def isTheMostImportantPiece(c: ChessPiece): Boolean = c match {
  case ChessPiece(_, King) => true
  case _ => false
}
```

Programação Funcional em Java ≥ 17

```
Object o = ...; // any object
String formatted = null;
if (o instanceof Integer i) {
    formatted = String.format("int %d", i);
} else if (o instanceof Long l) {
    formatted = String.format("long %d", l);
} else if (o instanceof Double d) {
    formatted = String.format("double %f", d);
} else {
    formatted = String.format("Object %s", o.toString());
}
```

```
Object o = ...; // any object
String formatter = switch(o) {
    case Integer i -> String.format("int %d", i);
    case Long l -> String.format("long %d", l);
    case Double d -> String.format("double %f", d);
    case Object o -> String.format("Object %s", o.toString());
}
```

Programação Funcional em JavaScript

```
const originalArray = [1, 2, 3];  
const newArray = [...originalArray, 4];
```

```
const person = { name: 'Alice', age: 30 };  
const updatedPerson = { ...person, age: 31 };
```

```
const add = (x, y) => x + y;  
const square = (x) => x * x;  
  
function compose(...functions) {  
  return (input) => functions.reduceRight((acc, fn) => fn(acc),  
input);  
}  
  
const addAndSquare = compose(square, add);  
  
console.log(addAndSquare(3, 4)); // 49
```

```
const numbers = [1, 2, 3, 4, 5, 6];  
  
const double = (num) => num * 2;  
const isEven = (num) => num % 2 === 0;  
  
const result = numbers  
  .map(double)  
  .filter(isEven)  
  .reduce((acc, num) => acc + num, 0);  
  
console.log(result); // 18
```


Programação Funcional (reativa) em ReactJS

```
const BooksList = () => {

  const [books, setBooks] = useState<Book[]>([])
  const [selected, setSelected] = useState<number | undefined>(undefined)

  const [inputTitle, searchTitle, setSearchTitle] = useInput("", "Search Title")
  const [inputAuthor, searchAuthor, setSearchAuthor] = useInput("", "Search Author")

  const loadBooks = () => {
    fetch("/books.json")
      .then(response => response.json())
      .then(data => setBooks(data))
  }
  useEffect(loadBooks, [])

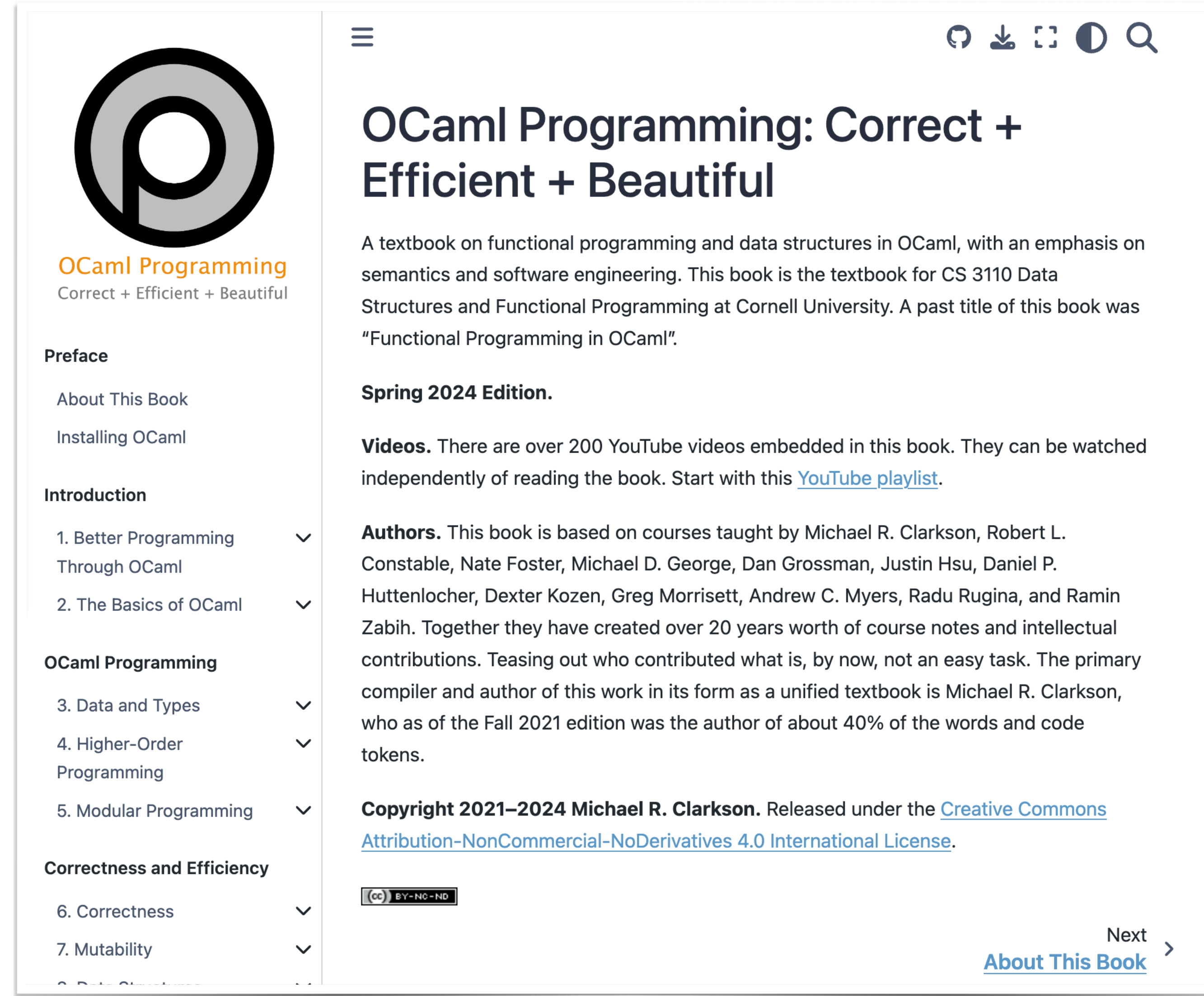
  const filteredBooks = books.filter(b => b.title.includes(searchTitle) && b.author.includes(searchAuthor))

  return <div>
    {inputAuthor}
    {inputTitle}
    <BooksListView books={filteredBooks} selected={selected} setSelected={setSelected} />
  </div>
}
```

Logística

Livro de texto

- OCaml Programming: Correct + Efficient + Beautiful
- Cornell University
- Michael R. Clarkson et al.
- Online resources (book, exercises, videos)



The screenshot shows the homepage of the OCaml Programming book. On the left is a sidebar with a table of contents. The main content area on the right features the book's title, a description, and details about the Spring 2024 edition, including a list of authors and the Creative Commons license.

OCaml Programming
Correct + Efficient + Beautiful

Preface

- About This Book
- Installing OCaml

Introduction

- 1. Better Programming Through OCaml ✓
- 2. The Basics of OCaml ✓

OCaml Programming

- 3. Data and Types ✓
- 4. Higher-Order Programming ✓
- 5. Modular Programming ✓

Correctness and Efficiency

- 6. Correctness ✓
- 7. Mutability ✓

OCaml Programming: Correct + Efficient + Beautiful


A textbook on functional programming and data structures in OCaml, with an emphasis on semantics and software engineering. This book is the textbook for CS 3110 Data Structures and Functional Programming at Cornell University. A past title of this book was "Functional Programming in OCaml".

Spring 2024 Edition.

Videos. There are over 200 YouTube videos embedded in this book. They can be watched independently of reading the book. Start with this [YouTube playlist](#).

Authors. This book is based on courses taught by Michael R. Clarkson, Robert L. Constable, Nate Foster, Michael D. George, Dan Grossman, Justin Hsu, Daniel P. Huttenlocher, Dexter Kozen, Greg Morrisett, Andrew C. Myers, Radu Rugina, and Ramin Zabih. Together they have created over 20 years worth of course notes and intellectual contributions. Teasing out who contributed what is, by now, not an easy task. The primary compiler and author of this work in its form as a unified textbook is Michael R. Clarkson, who as of the Fall 2021 edition was the author of about 40% of the words and code tokens.

Copyright 2021–2024 Michael R. Clarkson. Released under the [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).



Next [About This Book](#) >

Course Website & Discord

Lap-2025

Linguagens e Ambientes de Programação (2025 Edition)

(Para ler esta página em português, clique [aqui](#))

The curriculum of this course covers the fundamentals of programming language design, using functional programming in OCaml as a vehicle for a broader understanding of the concepts.

Corrections and suggestions are very much welcomed. (e-mail: [i.chirica <at> campus.fct.unl.pt](mailto:i.chirica@campus.fct.unl.pt))

Announcements

Important announcements and changes to this page will be posted in this section.

Referências

The primary reference for the course is the book “OCaml Programming: Correct + Efficient + Beautiful” by Michael R. Clarkson and others. The book is freely available online at OCaml Programming: [OCaml programming: Correct + Efficient + Beautiful](#). In addition to the course materials, instructors encourage reading the book and completing the exercises it proposes. The book also includes lecture videos from Cornell University’s CS3110 course, recorded during the pandemic.

During lectures, references will be made to materials from other books freely available online from the [Books On OCaml](#) page. Namely:

- “OCaml from the Very Beginning” by John Whittington. The book is freely available online at [OCaml From the Very Beginning](#).
- “Introduction to OCaml” by Jason Hickey. The book is freely available online at [Introduction to OCaml](#).
- “Real World OCaml” by Yaron Minsky, Anil Madhavapeddy, and Jason Hickey. The book is freely available online at [Real World OCaml](#).

Instructors Office Hours

Office hours will be held by appointment via email the day before at the following locations and contacts:

Teacher	Office hours	E-mail
João Costa Seco	Wednesday 17h (Gab. P2/1-II)	joao.seco <at> fct.unl.pt
Carla Ferreira		carla.ferreira <at> fct.unl.pt
Artur Miguel Dias		amd <at> fct.unl.pt
Ana Catarina Ribeiro	Tuesday 14h (TBD)	acm.ribeiro <at> campus.fct.unl.pt
Hugo Pereira	Wednesday 09h (TBD)	hg.pereira <at> campus.fct.unl.pt



Website



Discord

Ferramentas de laboratório

- Interpretador de OCaml + utop
- Compilador de OCaml: ocamlc + make/dune
- Trabalho com Visual Studio Code + OCaml Extension
- Exercícios com Jupyter Notebook + OCaml kernel + VSCode
- Trabalhos através do GitHub Classroom
- Correção automática dos trabalhos (mooshak ou outro equivalente)



Funcionamento

- Aulas teóricas com conceitos, discussão e exemplos.
- Aulas práticas com exercícios e apoio a projeto.

Programa

O plano tentativo para as aulas é o seguinte (pode haver mudanças a qualquer altura):

Semana	Data	Teóricas	Práticas	Materiais
1	6/3	Apresentação. Logística e avaliação. A história e o futuro das linguagens de programação.		
1	P		Não há aulas práticas	
2	10/3	Programação funcional. A linguagem OCaml. Expressões, Variáveis e tipos. Funções biblioteca. Input/Output básico.		
2	13/3	Declaração de nomes; declaração de funções, com e sem parâmetros; Avaliação de expressões por substituição; Funções como valores (primeira vez); Avaliação parcial de funções		
2	P		Kick the tyres: Instalação das ferramentas. OCaml, Jupyter, VSCode + plugin.	
3	17/3	Funções como valores. Composição. Polimorfismo. Inferência de tipos.		
3	20/3	Tipo função; Polimorfismo; Inferência de tipos.		
3	P			
4	24/3	Funções recursivas sobre naturais. Pensamento Indutivo vs. pensamento Iterativo.		
4	27/3	Tipos estruturados: produtos e registos. Exercícios.		
4	P			
5	31/3	Tipos estruturados: Listas e funções recursivas sobre listas. Exercícios.		

Avaliação

- Componente laboratorial
 - 3 projetos (P1', P2', P3')
 - D1, D2, D3 = Discussão P2, P3 (1h) - 30 de Maio
 - $P_i = \min(P_i', D_i)$
 - $\text{CompL} = 0.25 \times P1 + 0.25 \times P2 + 0.5 \times P3$, $\text{CompL} \geq 9.5$
- Componente teórico-prática
 - 1º Teste - T1 (1h30) - 16 de Abril
 - 2º Teste - T2 (1h30) - 30 de Maio
 - Exame - Ex (2h30) - ?
 - $\text{CompTP} = 0.4 \times T1 + 0.6 \times T2$ ou $\text{CompTP} = \text{Ex}$, $\text{CompTP} \geq 9.5$

Nota sobre Ferramentas de Inteligência Artificial

- As ferramentas de IA é permitida nos projetos e exercícios da aula e proibida nos testes escritos e discussões.
- O uso de ferramentas de IA têm de ser explicitamente referidas no código e relatório que for entregue.
- Considera-se que um aluno que use estas ferramentas durante a realização de um exercício de avaliação ou projeto e omita que as usou comete plágio.

“Analysis shows that, when prompted, 52 percent of ChatGPT answers to programming questions are incorrect and 77 percent are verbose. “