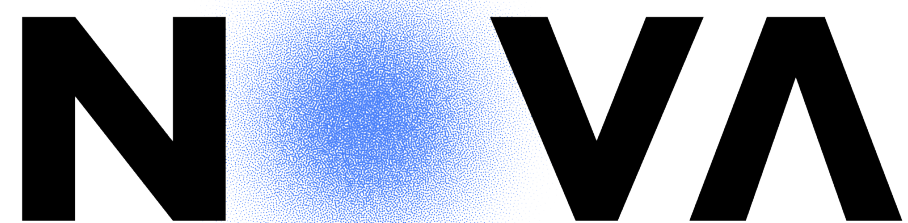


Programming Languages and Environments

LEI - Licenciatura em Engenharia Informática

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

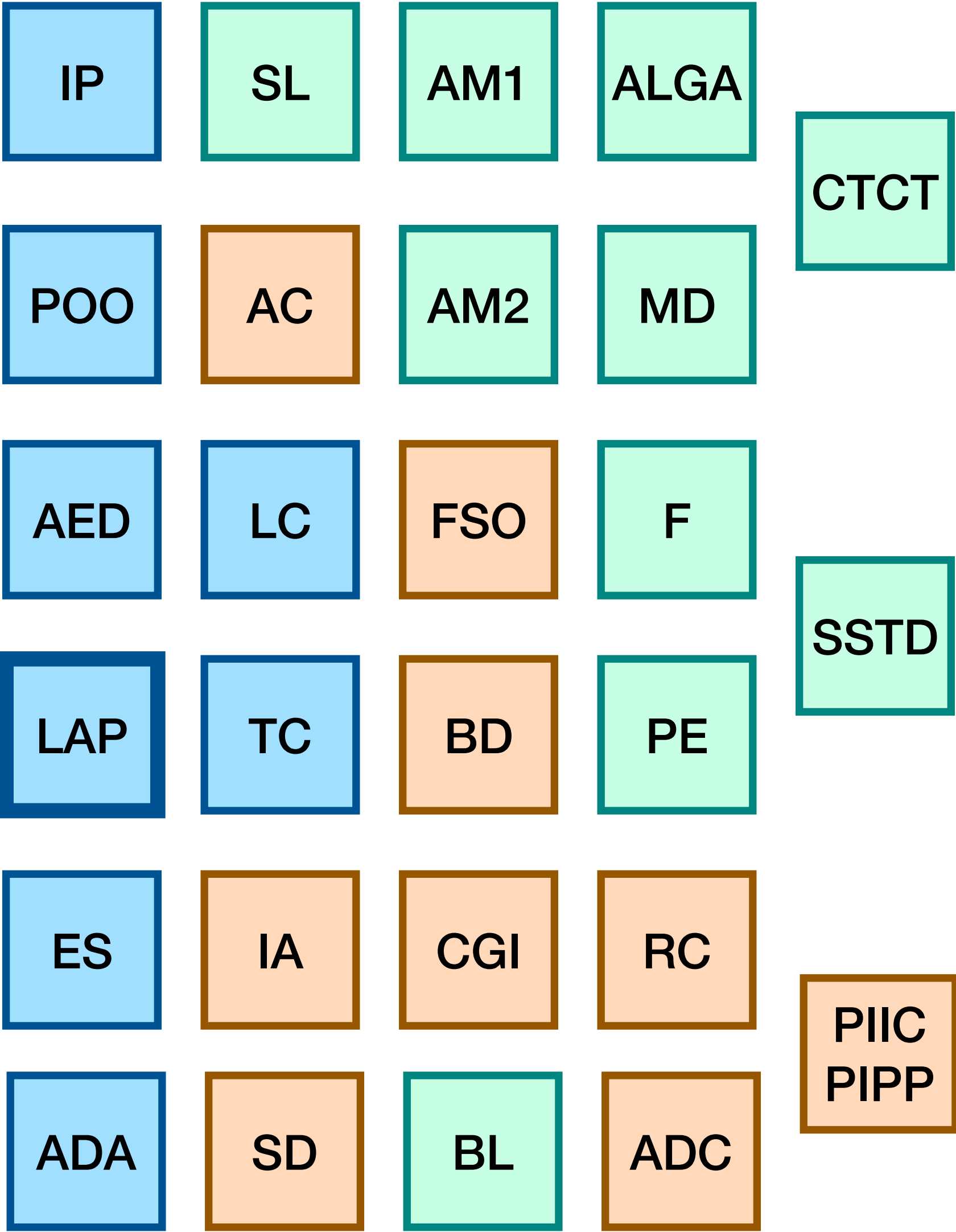


NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

LAP 2025

Curricular Unit

Contextualizing LAP



Programming Languages and Environments 2025

1. Study of a new paradigm: Functional Programming
2. Fundamentals of Programming Languages (PL)
3. Functional Programming is the communication medium for PLs
4. Functional Programming in OCaml
5. Learning multiple paradigms in a PL that does it best: compare and contrast

Why should you care about Programming Languages?

Why (1): Programming languages are the tool of the trade.

Why (2): Each language is more appropriate for a particular kind of problems. You need to know them deeply to be able to choose.

Why (3): Modern PLs have multiple paradigms. (it helps if you know them)

Why (4): PLs are precise models of reasoning.

Why (5): Designing PLs is more common than you think.

Why (6): If you know why, you can go further and make better.

Why should you care about Functional Programming?

Why (1): Functional PL have very strong semantics.

Why (2): Type systems enable strong reasoning.

Why (3): Non-interference. (e.g. name visibility, aliasing, side-effects, coercions).

Why (4): Understand and Devise other Languages.

Why (5): Immutability. (good programming languages do not need debuggers).

Why (6): Modularity and Composability.

Why (7): Higher-Order Abstraction. (genericity).

Why (8): Type safety. (absence of nulls, prevention of illegal operations).

Concepts

- Syntax, semantics and pragmatics
- Declarative and imperative languages
- Declaration and name definitions (binding, environment, scope)
- Parametric and universal polymorphism
- Abstractions and Applications
- Modularity and Composition
- Algebraic Data Types
- Type Systems (soundness) and Type Inference
- Reasoning and correctness (Testing vs Verification)
- Execution (Stack based)
- Lazy vs Strict
- Persistent Data Structures
- Pitfalls: state, aliasing, etc.
- Interpreters and Compilers (Just-in-time)

Functional Programming in 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));
```


Functional Programming in 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);
}
```

Functional Programming in 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
}
```

Functional Programming in 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());
}
```

Functional Programming in 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
```


Functional (Reactive) Programming in 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>
}
```

Logistics

Course textbook

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



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 ✓

8. Data Structures ✓



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

Laboratory Tools

- OCaml Interpreter + utop
- OCaml Compiler: ocamlc + make/dune
- Projects in Visual Studio Code + OCaml Extension
- Exercises/course work in Jupyter Notebook + OCaml kernel + VSCode
- Project delivery via GitHub Classroom
- Automated correction of project results (dune unit tests)



Methodology

- Lectures with concepts, discussion, and examples.
- Practical classes with exercises and project support.

Syllabus

The plan for the theoretical classes is as follows: (This may change anytime without warning):

(The dates for the english version correspond to the lecture T1, and lab P1.)

Week	Date	Theoretical	Practical	Materials
1	5/3	Introduction. Logistics and evaluation. The history and future of programming languages.		
1	P		No labs	
2	11/3	Functional programming. The OCaml language. Expressions, Variables, and Types. Library functions. Basic Input/Output.		
2	12/3	Name declaration; function declaration, with and without parameters; Expression evaluation by substitution; Functions as values (first time); Partial function evaluation.		
2	P		Kick the tyres: Tool installation. OCaml, Jupyter, VSCode + plugin.	
3	18/3	Functions as values. Composition. Polymorphism. Type inference.		
3	19/3	Function type; Polymorphism; Type inference.		
3	P			
4	25/3	Recursive functions on natural numbers. Inductive vs. Iterative thinking.		
4	26/3	Structured types: products and records. Exercises.		
4	P			
5	1/4	Structured types: Lists and recursive functions on lists. Exercises.		
5	2/4	Structured types: Higher-order programming: map and fold. Exercises. Presentation of the First Assignment.		
5	P			

Evaluation

- Practical component
 - 3 projects (P1', P2', P3')
 - D1, D2, D3 - Discussion P1, P2, P3 (1h) - May 30th
 - $P_i = \min(P_i', D_i)$
 - $\text{CompL} = 0.25 \times P1 + 0.25 \times P2 + 0.5 \times P3, \text{CompL} \geq 9.5$
- Theoretical component
 - Midterm test - T1 (1h30) - April 16th
 - Final test - T2 (1h30) - May 30th
 - Exam - Ex (2h30) - ?
 - $\text{CompTP} = 0.4 \times T1 + 0.6 \times T2$ or $\text{CompTP} = \text{Ex}, \text{CompTP} \geq 9.5$

On the usage of Artificial Intelligence

- The usage of AI tools is allowed on the development of the project and lab exercises. But it is strictly forbidden during tests/exam and discussions.
- The use of AI tools needs to be explicitly disclosed in the code and report that is to be submitted. Failing to do so counts as plagiarism.

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