

Universidade da Beira Interior

# Programar em OCaml Introdução pela prática

Simão Melo de Sousa

Aula 0 - O que deve saber antes de começar

- The name of the game
- Um pouco de história
- Emergência de OCaml
- Quem usa OCaml?
- A comunidade
- OCaml numa imagem
- Ambiente de trabalho OCaml
- Primeiros passos em OCaml

Estes acetatos foram preparados com base no material pedagógico que encontrará no curso MOOC:

## **Introduction to Functional Programming in OCaml**

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

-

Week 0

que começa este ano a dia 26 de Setembro (curso de 6 semanas)

---

## The name of the game

# Da Elegância e Simplicidade em Programação

*Beauty is more important in computing than anywhere else in technology because software is so complicated. **Beauty is the ultimate defence against complexity.***

— David Gelernter

*When I am working on a problem I never think about beauty. I think only how to solve the problem. But when I have finished, **if the solution is not beautiful, I know it is wrong.***

— R. Buckminster Fuller

*Complexity has nothing to do with intelligence, **simplicity does.***

— Larry Bossidy

***Simplicity** is prerequisite for **reliability.***

— Edsger W. Dijkstra

*Sometimes, **the elegant implementation is a function.**  
Not a method, not a class, not a framework. Just a function.*  
— John Carmack

Estas aulas visam introduzir a programação funcional que em muito está alinhada com estes princípios aqui expostos.

Para este propósito, vamos usar a linguagem *state-of-the-art* **Objective Caml** (OCaml)

Mas nunca se esqueça:

**Não existam boas linguagens, só existem bons programadores!**

dado um desafio, um bom programador **sabe escolher** a melhor ferramenta (linguagem de programação, API, etc.) e a melhor resolução (algoritmos, estrutura de dados, etc.)

como veremos, as linguagens de programação têm os seus domínios de excelência

---

## História e motivação



**Computing:** *the study of algorithmic processes that describes and transform information. The fundamental question is “what can be (efficiently) automated?” — 1989 ACM report on Computing as a Discipline*

as componentes básicas da *computação*

- um **programa** que descreve as transformações que desejamos aplicar à informação por processar
- uma **máquina** que seja capaz de executar tais transformações

há **muitas** máquinas e **muitas formas** possíveis de escrever programas algumas até foram inventadas antes do primeiro computador!

visitemos um pouco a história da computação para melhor contextualizar linguagens como OCaml

os detalhes desta história serão abordados na UC de Teoria da Computação (TC)

em 1928, David Hilbert, confiante na reconstrução da matemática operada na altura, enunciou o desafio seguinte:

**Problema da Decisão:** Poderemos definir um **processo** que possa determinar num **número finito** de **operações** se uma determinada fórmula lógica (predicativa de primeira ordem) é verdade ou falsa?



este desafio pertence ao conjunto dos actos fundadores da informática. Mais será dito em TC, mas em síntese o desafio pergunta se existem um mecanismo (automático) que possa revolver genericamente qualquer problemas matemático

no fundo:

**podemos resumir a matemática a algo que virá a chamar-se de informática?**

**Problema da Decisão:** Poderemos definir um **processo** que possa determinar num **número finito** de **operações** se uma determinada fórmula lógica é verdade ou falsa?



**podemos resumir a matemática a algo que virá a chamar-se de informática?**

Rapidamente a resposta veio a ser conhecida, e não foi positiva. Foi **duplamente negativa**

**propriedade de correcção:** o que se estabelece, está sempre certo

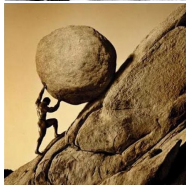
**propriedade de completude:** consegue-se sempre estabelecer uma conclusão

**Teoremas de Incompletude:** nenhum sistema axiomático (e.g. lógico) não trivial pode ser simultaneamente completo e correcto. Para garantir a correcção (o que é fundamental) é preciso abdicar da completude.



assim

**a Matemática é um edifício em perpétua construção e remodelação**



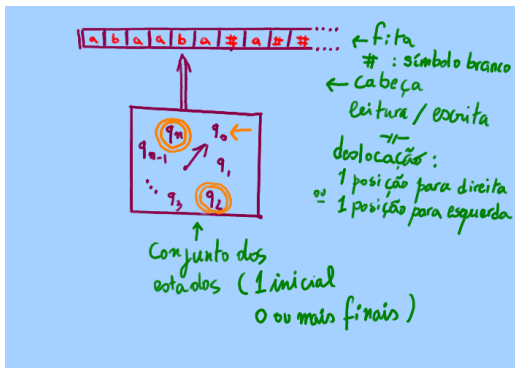
em jeito de curiosidade, a auto-referência (e.g. recursividade) e o paradoxo do mentiroso têm o seu papel nesta história

percebemos que a procura da verdade absoluta é um caminho sem fim à vista

mas se aceitarmos jogar em terrenos menos vastos do que a matemática na sua generalidade e considerar teorias correctas e completas (mas menos abrangentes) ou então abdicar da completude, podemos reduzir estas matemáticas a um processo mecânico automatizável?

Para responder a este desafio é preciso definir com rigor o que entendemos por **operações** e **processo de cálculo** (o que é a informática?)  
Turing e Church propuseram definições para estes conceitos e deram uma segunda resposta negativa ao desafio

inventou o conceito de Máquina de Turing



estas máquinas dão um fundamento teórico às arquitecturas de computadores e à programação imperativa

a fita tem o papel da **memória** (onde se encontram dados e programas)  
o autómato o do **processador**

# Máquinas de Turing e o paradigma imperativo

Um programa imperativo **lê, escreve, executa operações** e **toma decisões** com base no conteúdo de células de memória que contêm informação sobre as variáveis do programa, como `x,n,res` no seguinte programa java (que calcula o factorial)

```
public class Factorial
{
    public static int compute(int n)
    {
        int res = 1;
        for (int c = 1; c <= n; c++)
            res = res * c;
        return res;
    }
}
```

tal como Alan Turing, e no mesmo ano de 1936, Alonzo Church (orientador do A. Turing) dá uma definição alternativa de algoritmo e da execução de programas responde pela negativa à questão de D. Hilbert

a sua definição é de natureza bem diferente da das máquinas de Turing, embora se descobre cedo que ambas são equivalentes  
A. Church introduz o **cálculo lambda** ( $\lambda$ -calculus)

- Elemento de base:  $x$ , variável.
- Abstracção:  $\lambda x.M$ , função anónima de um parâmetro formal,  $x$  e de corpo  $M$
- Aplicação:  $(M N)$ , a função  $M$  aplicada ao parâmetro efectivo  $N$



Computação: a regra da redução  $\beta$ :  $(\lambda x.M)N \rightarrow_{\beta} M[x := N]$

**programa:** um termo do cálculo lambda

**execução:** **basta a aplicação** da regra  $\beta$  (nada de hardware complexo)



# o cálculo lambda e a programação funcional

num **programa funcional** definimos **funções** (possivelmente recursivas), **compomo-las** e **aplicamo-las** para **calcular** os **resultados** esperados

tal como no seguinte exemplo

```
let rec fact =  
  function n -> if n=0 then 1 else n * (fact (n-1))
```

numa linguagem de programação funcional pura, as funções são cidadãs de primeira classe, como qualquer outro valor (como os inteiros, por exemplo) podem

- lhes ser atribuído um nome (ou não)
- ser avaliadas
- ser passadas como argumento (de outras funções)
- ser devolvidas como resultado de funções
- ser usadas em qualquer local onde se espera uma expressão

veremos a importância destes factos nas lições que se seguem

(ab)usando da notação original de A. Church podemos reescrever o código seguinte

```
function n -> if n=0 then 1 else n * (fact (n-1))
```

na forma

$$\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n * (\text{fact } (n - 1))$$

isso são as agora famosas **lambda expressions** que se introduziram no Java e C++.

São elementos basilares da programação funcional que as linguagens *mainstream* agora integraram

em 1937, A. Turing demonstrou que o cálculo lambda e as máquinas de Turing são formalismos com a mesma expressividade: uma função é **computável** por uma máquina de Turing **se e só se** é **computável** no cálculo- $\lambda$

a **Tese de Church-Turing** foi então formulada e afirma que uma função **computável** (um algoritmo) em **qualquer formalismo computacional** é também **computável por uma máquina de Turing**

em termos mais leigos:

- todas as linguagens de programação são **computacionalmente equivalentes**
- **qualquer algoritmo** pode ser expresso usando um destes formalismos (que são equivalentes)

## Mas as linguagens de programação **não nascem todas iguais...**

são computacionalmente equivalentes, mas tem comodidades e expressividade diferentes

a procura das construções programáticas mais expressivas ou mais cómodas é uma luta sem fim que leva a

- diferentes formas de representação dos dados
- diferentes modelos de execução
- diferentes mecanismos de abstracção

e muitas outras características desejáveis entram em conta no momento do desenho de novas linguagens de programação

- segurança da execução
- eficiência
- modularidade e capacidade de manutenção
- etc.

**dependendo do problema em causa, algumas linguagens de programação serão mais adequadas do que outras**

# uma opinião do criador da linguagem FORTRAN

... e precursor da programação funcional

*Functional programs deal with structured data, ... do not name their arguments, and do not require the complex machinery of procedure declarations ...*

***Can programming be liberated from the von Neumann style?***

— John Backus, Turing lecture, 1978



# Porque a programação funcional está para ficar?

uma reflexão sobre o ensino introdutório em informática na CMU (<http://www.cs.cmu.edu/~bryant/pubdir/cmu-cs-10-140.pdf>) destaca tendências emergentes das quais:

*Need for greater software reliability.*

*(Pure) functional programs are easier to prove correct than imperative ones*

*Harnessing the power of parallel computation.*

*A carefully chosen set of higher order functions allows to write programs that are easily parallelisable.*

um exemplo bem conhecido que advém directamente da programação funcional é o paradigma cloud **MapReduce**

a **programação funcional** é longe de ser um exclusivo das linguagens de programação ditas funcionais, **pratica-se em todas as linguagens**

as suas vantagens estão largamente reconhecidas  
clareza, elegância, simplicidade e expressividade:

- a **recursividade** é uma componente habitual de qualquer linguagem de programação
- Java 1.5 e C#/.NET introduziram os **genéricos** (polimorfismo de tipos, nas linguagens funcionais tipadas)
- Java 1.8 e C++ (v. 11) introduziram as **lambda expressions** (as funções puras, como já as descrevemos) e as construções relacionadas (fluxos)
- etc.

a **programação funcional** é longe de ser um exclusivo das linguagens de programação ditas funcionais, **pratica-se em todas as linguagens**

recentes linguagens de programação **assimilam** nas suas construções as **boas práticas** que as linguagens de programação funcionais estabeleceram desde sempre  
citemos linguagens como **rust**, **swift**, **scala**, **python**, **typescript**, **go**, **closure** (que é funcional por definição) que muito devem à programação funcional

**mensagem por memorizar**: quaisquer que sejam as vossas linguagens de programação favoritas, perceber os princípios da programação funcional

- **é uma competência transversal, importante e de base**
- **torna-vos sem dúvida melhores programadores**

exemplos de linguagens funcionais: **Haskell**, **Racket**, **Scheme**, **SML**, etc. Mas propomos fazer esta viagem nas linguagens de programação funcionais com **OCaml**.



---

## Da emergência de OCaml

## A Theory of Type Polymorphism in Programming

ROBIN MILNER

*Computer Science Department, University of Edinburgh, Edinburgh, Scotland*

Received October 10, 1977; revised April 19, 1978

The aim of this work is largely a practical one. A widely employed style of programming, particularly in structure-processing languages which impose no discipline of types, entails defining procedures which work well on objects of a wide variety. We present a formal type discipline for such polymorphic procedures in the context of a simple programming language, and a compile time type-checking algorithm  $\mathcal{M}$  which enforces the discipline. A Semantic Soundness Theorem (based on a formal semantics for the language) states that well-typed programs cannot "go wrong" and a Syntactic Soundness Theorem states that if  $\mathcal{M}$  accepts a program then it is well typed. We also discuss extending these results to richer languages; a type-checking algorithm based on  $\mathcal{M}$  is in fact already implemented and working, for the metalanguage ML in the Edinburgh LCF system.

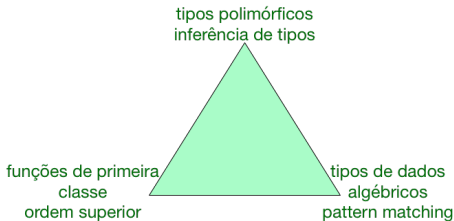
### 1. INTRODUCTION

The aim of this work is largely a practical one. A widely employed style of programming, particularly in structure-processing languages which impose no discipline of types (LISP is a perfect example), entails defining procedures which work well on objects of a wide variety (e.g., on lists of atoms, integers, or lists). Such flexibility is almost essential

OCaml pertence a família das linguagens funcionais **fortemente tipadas**, com **tipagem estática**

esta família iniciou-se com o trabalho de Sir Robin Milner (na linguagem **ML**)

# As características da família de linguagens ML



- **tipagem estática forte:** qualquer expressão da linguagem rigorosamente tem um tipo, verificado em tempo de compilação
- **inferência de tipos:** mas não é preciso defini-lo, o compilador sabe calculá-lo
- **pattern-matching:** é possível questionar a estrutura de um dado e usá-la

1980: o projecto Formal no INRIA, na direcção de Gérard Huet

- trabalho pioneiro em mecanização da matemática
- utiliza originalmente a linguagem ML de Milner
- neste contexto faz contribuições importantes à linguagem (em particular o mecanismo de *pattern-matching*)
- resolve criar a sua própria versão de ML

emergência do primeiro Caml

- 1985: Guy Cousineau, Pierre-Louis Curien e Michel Mauny criam a **Categorical Abstract Machine** (CAM)
- 1987: Ascender Suarez publica a primeira implementação de Caml
- 1988-1992: Michel Mauny e Pierre Weis alteram e estendam o Caml

nesta fase a linguagem ganhou algum estatuto na comunidade académica mas era complexa e necessitava de computadores de alto desempenho (da época)

início da década de 90: a era do Caml-light

1990-1991: Xavier Leroy cria a **máquina abstracta Zinc**, Damien Doligez implementa um gestor de memória otimizado e estes esforços resultam no **Caml-light**

- cabe numa disquete (binário de pequeno tamanho)
- portátil: suportado num interpretador de bytecode
- eficiente: pode ser executado em computadores pessoais

a máquina Zinc era conceptualmente muito diferente da CAM, mas o nome ficou

- 1995: **Caml Special Light**: compilador nativo, sistema de módulos
- 1996: **Objective Caml**: camada objecto eficiente e elegante (Jérôme Vouillon e Didier Remy)
- 2000: Integração de métodos polimórficos, *labels*, argumentos opcionais, variantes polimórficos (Jacques Garrigue)
- 2011: o nome da linguagem estabiliza-se para **OCaml**

ao longo dos anos OCaml ganhou maturidade e providencia agora um conjunto único e expressivo de mecanismos programáticos.

---

## Quem usa OCaml?

OCaml não é uma linguagem *mainstream*, mas é uma linguagem de programação com uma larga comunidade de utilizadores e de disseminação alargada na academia.

**Portugal:** aqui na UBI, na FCT-UNL, U. Évora

**França:** Univ. Paris Diderot, Pierre et Marie Curie, Paris Saclay, Rennes, École Normale Supérieure, École Polytechnique etc.

**Europa:** Univ. de Pisa, Bologna, Birmingham, Cambridge, Aarhus, Innsbruck, Varsóvia, etc.

**Estados Unidos:** Cornell, Harvard, MIT, Pennsylvania, Princeton, etc.

e muitos outros...

(exemplos de) projectos de investigação *top-notch*:

- **Sistema de prova Coq** (ACM Software System Award 2014)
- **Analizador estático Astrée** (verifica o software do Airbus A380)
- **Plataforma de análise e verificação Frama-C** (análise de programas C)
- **Ocsigen** (framework rica e completa para aplicações web)
- **Alt-Ergo** (SMT solver de ponta)
- **Mirage OS** (kernel/Sistema Operativo)
- **Flow / Hack** (*type checkers* para PHP/Javascript)
- **Infer** (analizador estático de aplicações móveis - Android e iOS)

e muitos outros ....



- Bloomberg, finance
- Citrix, virtualisation, cloud
- Dassault, aerospace
- Facebook
- JaneStreet Capital, finance
- LexiFi, finance
- Google
- Microsoft
- RedHat
- Cryptosense
- Trust-in-Soft

damos em seguida alguns testemunhos

# garantir a segurança de código em sistemas embutidos críticos: Astrée

Astrée é o analisador estático usado pela Airbus para provar a ausência de bugs nos sistemas de comando e controlo do A380.

*A **type-safe functional language** was the natural choice to implement the Astrée analyzer. OCaml's robust design supported a scalable development process, from research to industry, and we appreciated its **high performance native code compiler**.*

— Antoine Miné, Researcher at CNRS & ENS (2015)

Os sistema de prova COQ é uma sistema de prova de referência mundial utilizado em provas por computador complexas. Por exemplo, realizaram-se com o seu auxílio, provas de seguranças complexas (**Smart Cards**, sistemas criptográficos, etc.)

*Amongst all the great features of OCaml, **pattern matching** is crucial for Coq: without it, implementing complex symbolic computations would be a nightmare!*

*— The Coq development team (2015)*

TrustInSoft fornece soluções inovadoras de software de segurança (*security and safety*)

*OCaml generates code that's **very efficient** compared to other languages with similar expressivity. **Expressivity** is needed when developing sophisticated static analyzers. **Efficiency** is necessary when working at the frontier of what is possible at all on today's computers. **Static typing** saves clock cycles at execution time and, more importantly, human time during development.*

— Pascal Cuoq, TrustInSoft (2015)

o projeto Ocsigen permite a escrita de aplicações web avançadas e *state-of-the-art*

*OCaml's type system allows Ocsigen to **check statically advanced properties** of a Web application, like ensuring that a program will **never generate invalid HTML pages**, or that **a form has the expected fields**.*

*The advantages of this powerful type system become obvious when **refactoring a large project**: the compiler points out every piece of code that needs to be modified, **saving days of testing and debugging**.*

— Vincent Balat, criador do Ocsigen (2015)

OCamlPro é especialista em soluções de desenvolvimento no ecossistema OCaml

*I have tried many programming languages, but none of them could compete with OCaml. In OCaml, you just **define the type of your data**, and the compiler will gently **drive you towards your destination**, at highspeed on a highway. It's just fascinating!*

— Fabrice Le Fessant, OCamlPro (2015)

Cryptosense desenvolve soluções de software para a auditoria de segurança em criptografia

*We see OCaml as a strategic advantage. It helps us to **rapidly** produce **high-quality readable, reusable code**, which is essential for a start-up.*

*— Graham Steel, Cryptosense (2015)*

LexiFi desenvolve software inovador na área da finança que permite a gestão de produtos financeiros complexos, combinando computação numérica com computação simbólica avançada

***Safety, readability, expressivity and great performance** are often cited as key benefits of OCaml. We also value the **portability** of the system, as our products are deployed on Unix, Windows and **in the web browser**. Parts of our codebase which were historically written in C, C# or Javascript are now in OCaml. As one of the earliest industrial adopters of OCaml, we are delighted to observe the **growing interest and activity** around OCaml in the last years.*

— Alain Frisch, LexiFi (2015)



Citrix e a Universidade de Cambridge desenvolvem actualmente o sistema operativo Mirage, um *exokernel* para Xen integralmente desenvolvido em OCaml

*OCaml's combination of static type safety and fast native code compilation has been essential to our MirageOS project, which rebuilds operating system components (including TCP/IP and device drivers) in a safe, modular and flexible style.*

— Anil Madhavapeddy, Universidade de Cambridge (2015)

JaneStreet utiliza OCaml para a construção de ferramentas de *financial trading* capaz de lidar com transações de alta velocidade de mais de 10 mil milhões de dollars por dia

*Our experience with OCaml on the research side convinced us that we could build **smaller, simpler, easier-to-understand systems** in OCaml than we could in languages such as Java or C#. **For an organization that valued readability, this was a huge win...***

*There is, a surprisingly wide swath of bugs against which **the type system is effective**, including **many bugs that are quite hard to get at through testing.***

— Yaron Minsky. *Em OCaml for the masses.*  
*Communications of the ACM, September 2011*

Xen é um *hypervisor* que alimenta milhões de máquinas virtuais na nuvem. As ferramentas de gestão e controlo são desenvolvidas em OCaml

*OCaml has brought **significant productivity and efficiency benefits** to the project. OCaml has enabled our engineers to be more productive than they would have been had they adopted any of the mainstream languages.*

— Richard Sharp, Citrix

# Em resumo, o que dizem do uso de OCaml

Há uma variedade notável de utilizadores e de utilização de OCaml. De forma unânime valorizam:

segurança

da **tipagem forte estática** ao *pattern matching*

eficiência

um **compilador de alta performance**

expressividade

combinação de uma **linguagem funcional** com **inferência de tipos** e **polimorfismo**

---

## OCaml numa imagem

um conjunto rico de ferramentas de desenvolvimento

- **opam**: um gestor de pacotes de software (*package manager*) para a instalação facilidade (com gestão de dependências) de ferramentas e bibliotecas de que necessita  
ver <http://opam.ocaml.org>
- centenas de bibliotecas e ferramentas disponíveis  
ver <http://opam.ocaml.org/packages>
- debugger
- profiler
- unit testing, etc.

um conjunto rico de ferramentas de compilação

- `ocaml` um REPL (Read-Evaluate-Print-Loop), para o **desenvolvimento rápido**
- `ocamlc` um compilador para bytecode, **para código portátil** (ver <http://caml.inria.fr/ocaml/portability.en.html>)
- `ocamlopt` um compilador nativo (AMD64, IA32, Power PC, ARM) para **executáveis otimizados**
- `js_of_ocaml` compilador para JavaScript, para construir **aplicações web**

Ocaml tem um `toplevel` plenamente funcional que

- **lê** o programa, frase por frase
- **compila**, cada frase lida, *on-the-fly*, reporta erros se encontrados
- **avalia** a frase se está tudo correcto
- **mostra** o resultado

isto significa que consegue **ver os resultados** do seu programa **a medida que o escreve** sem necessitar que escreva uma interface com o utilizador



# Uma primeira experiência com o OCaml Toplevel

```
> ocaml
      OCaml version 4.03.0

# print_string "Olá Mundo!\n";;
Olá Mundo!
- : unit = ()

# 1+4;;
- : int = 5

# List.map (fun x-> x+1) [1;2;3;4;5;6;7];;
- : int list = [2; 3; 4; 5; 6; 7; 8]

# let x = 3 * 9;;
val x : int = 27

# let rec fact n = if n < 1 then 1 else n * (fact (n-1));;
val fact : int -> int = <fun>

# fact 5;;
- : int = 120
```

podemos experimentar programas OCaml **directamente no browser**

<https://try.ocamlpro.com/>

usufruir de OCaml sem nenhuma instalação! como?

- o toplevel OCaml em causa é escrito em OCaml
- foi compilado para bytecode com `ocamlc`
- foi compilado para JavaScript com `js_of_ocaml`
- carregado para o browser no momento do acesso ao site

## Try OCaml

OCaml is a strongly typed functional language. It is concise and fast, enabling you to improve your coding efficiency while producing code with higher quality.

Type `!lesson 1` to start the tutorial.  
(click on the code to insert)

English

Lesson 1	Simple Expressions
Lesson 2	Imperative Programming
Lesson 3	Functions
Lesson 4	Pattern Matching
Lesson 5	Syntax Trips
Lesson 19	The Graphics module
Lesson 20	New Features in 3.12.1

### Connect to OCamlPro

If you want to know more about OCaml, and how Functional Programming can boost software projects in your company, contact us at OCamlPro. We provide on-site training from beginners to experts, and our highly-experienced experts will always give insightful advises on your ongoing projects.



Send Clear Reset Save

Commands	Effects
Enter / Return	Submit code
Up / Down	Cycle through history
Shift + Enter	Multiline edition
<code>!lesson 1</code>	Move to lesson 1
<code>step 1</code>	Move to step 1 of the current lesson
<code>lessons</code>	See available lessons
<code>steps</code>	See available steps in the current lesson

All Try OCaml sites:

---

## Ambiente de trabalho Ocaml: Instalação e utilização

site de referência: [www.ocaml.org](http://www.ocaml.org)

OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles

[Install OCaml](#)

[en fr]

### Learn

Find out about OCaml, read about users, see code examples, go through tutorials and more.

### Documentation

Install OCaml, look up package docs, access the Manual, get the cheat sheets and more.

### Community

Read the news feed, join the mailing lists, get support, attend meetings, and find OCaml around the web.

### News

- MOOC OCaml (Online Course) October 19, 2015
- OCaml 2016 September 23, 2016
- Weekly News July 5, 2016
- The first version of the Frama-Clang plugin... July 4, 2016
- Minimising the virtual machine monitor July 2, 2016
- Behavioural types June 30, 2016

[More...](#)

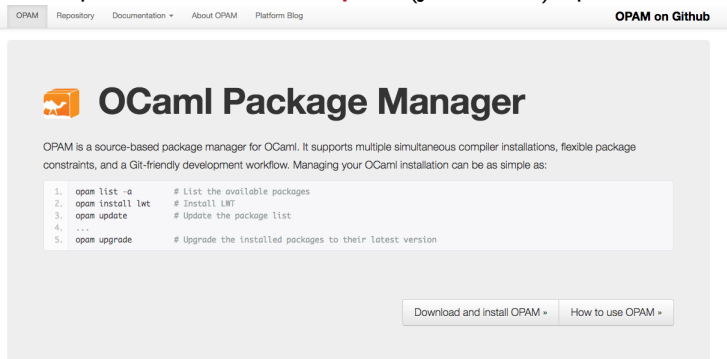
### A taste of OCaml

```
(* Binary tree with leaves carrying an integer. *)  
type tree = Leaf of int | Node of tree * tree
```

### Packages

Package	Version	Date
---------	---------	------

gestor de pacotes de software **opam** (já referido) [opam.ocaml.org/](http://opam.ocaml.org/)



The screenshot shows the OPAM website homepage. At the top, there is a navigation bar with links for "OPAM", "Repository", "Documentation", "About OPAM", and "Platform Blog". On the right side of the navigation bar, it says "OPAM on Github". The main content area features the OCaml logo (an orange box with a white silhouette of a camel) and the title "OCaml Package Manager". Below the title, a paragraph describes OPAM as a source-based package manager for OCaml, supporting multiple simultaneous compiler installations, flexible package constraints, and a Git-friendly development workflow. A code block lists five basic commands: 1. `opam list -a` (List the available packages), 2. `opam install lwt` (Install LWT), 3. `opam update` (Update the package list), 4. `...`, and 5. `opam upgrade` (Upgrade the installed packages to their latest version). At the bottom of the main content area, there are two buttons: "Download and install OPAM" and "How to use OPAM".

## News

## Contribute

- 👉 Report bugs and ask for features in the OPAM tool
- 👉 Report packaging issues or request new packages
- 👉 Address general queries on the tool and packages
- 👉 Discuss the tool internals

## Tutorials

- Installing OPAM
- Creating Packages
- Developer Manual

**1239** packages

### New packages

lacaml 9.1.0

Jul 6

### Most Downloaded Packages (this month)

ocamlfind

10927

**IDE completo:** (`gnu`) `emacs` ou `aquamacs` ou `vim` em conjunto com o modo `tuareg` (coloração de sintaxe e IDE), `ocp-indent` (identação), `merlin` (add-ons do IDE) (ver link: [turn-your-editor-into-an-ocaml-ide](#))

`ocaide` um plug-in Eclipse para OCaml link: [www.algo-prog.info/ocaide/](http://www.algo-prog.info/ocaide/) pode em alternativa usar o `Sublime Text Editor` que suporta a sintaxe OCaml

`ocamldoc` (vem com o compilador) ou `ocamlweb` (link) para gerar documentação a partir do código fonte (como o javadoc)

`ocamldebug` para depurar (*debug*) código (bytecode) ocaml (vem com o compilador). Poderá usar o gdb ou ddd para código compilado nativamente

`ocamlprof` para fazer profiling em código OCaml(vem com o compilador)

`ocaml-metrics` para calcular métricas de qualidade de código de programas OCaml (link) - à semelhança do *codacy*

`ocamlbuild` gestor de compilação de projectos OCaml (disponível via `opam`) ou o makefile générico para projectos OCaml `ocaml-makefile` (disponível igualmente via `opam`)

`kaputt` plataforma de teste unitário para projectos ocaml (link: [kaputt.x9c.fr](http://kaputt.x9c.fr))

para principiantes: [ocaml\\_beginners@yahoo.com](mailto:ocaml_beginners@yahoo.com)

para utilizadores experientes: [caml-list@inria.fr](mailto:caml-list@inria.fr)

---

## primeiros passos em OCaml



vamos apresentar rapidamente algumas características da linguagem OCaml antes da apresentação detalhada que segue nas lições seguintes

a unidade básica programática em OCaml é a **expressão**

à diferença de C, não há diferença entre expressão e instrução  
estas são a mesma coisa em OCaml

um **programa** OCaml é assim **uma sequência de expressões**

à diferença de C onde há um ponto de entrada (a função `main`), um programa é  
um conjunto de expressões e nenhuma tem um estatuto especial

a **execução** de um programa consiste em **calcular o valor das expressões** da  
primeira à última

a execução de uma expressão resulta num **valor**

## **regra de ouro:**

- **cada expressão tem um valor**
- **cada expressão tem um e um só tipo**

OCaml providencia nativamente a estrutura de dados das listas (ligadas): o tipo **list**

uma lista ou é a lista vazia (notação: `[]`) ou é a lista que tem um elemento (digamos `x`) à cabeça e dispõe de uma cauda (digamos `l`)  
a cauda é o resto da lista  
neste último caso a notação é `x::l`

comparando com a linguagem C, a lista vazia `[]` é o apontador `null`

quando olhamos para a lista `x::l`, `x` é o conteúdo da célula inicial e `l` é para onde aponta o apontador `next`, logo é também uma lista

as listas são **polimórficas**: conceptualmente, podem conter qualquer tipo de elementos, mas se um elemento é de um determinado tipo, então todos os outros elementos são também deste tipo

**exemplo:** `1::2::3::[]` (ou mais comodamente `[1;2;3]`)

vamos escrever uma função que soma os valores de uma lista de inteiros

```
let rec suml = function
  []          -> 0
| a::cauda   -> a + (suml cauda)
```

em nenhum momento nesta definição, indicamos os tipos dos parâmetros e variáveis. Ora, pelas **regras de ouro**, qualquer expressão tem um tipo

```
val suml : int list -> int = <fun>
```

o **type checker** OCaml **infere automaticamente** (i.e. calcula) o tipo certo em nosso lugar, sem custo!

## Well-typed programs cannot go wrong (Robin Milner)

todos os tipos são **calculados** em *tempo de compilação*  
a disciplina de tipos correspondente é igualmente **assegurada** em tempo de compilação

```
suml [1;2;3];;
```

```
- : int = 6
```

mas

```
["1";"2";"3"];;
```

```
- : string list = ["1"; "2"; "3"]
```

```
suml ["1";"2";"3"];;
```

Characters 6-9:

```
suml ["1";"2";"3"];;
```

```
^^^
```

```
Error : This expression has type string but an expression was expected  
of type int
```

podemos generalizar a nossa função de soma sobre listas de inteiros

```
let rec suml = function
  [] -> 0
| a::rest -> a + (suml rest);;
```

```
let rec fold op e = function
  [] -> e
| a::rest -> op a (fold op e rest);;
```

novamente, sem precisar de indicação nossa no programa, o type-checker de OCaml infere o tipo **mais geral**

```
val fold : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b = <fun>
```

```
fold ( + ) 0 [1;2;3;4;5];;
```

```
- : int = 15
```

```
fold ( * ) 1 [1;2;3;4;5];;
```

```
- : int = 120
```

```
fold ( ^ ) ["1";"2";"3"];
```

```
- : string = "123"
```

```
fold ( fun (x,y) a -> x + a ) 0 [(2,4);(3,5)];;
```

```
- : int = 5
```

imaginemos que queiramos escrever uma função que, numa lista de elementos, elimine os que são consecutivamente repetidos

```
let rec destutter = function
  | []          -> []
  | x :: y :: rest ->
      if x = y then destutter (y :: rest)
      else x :: destutter (y :: rest) ;;
```

Warning 8: this pattern-matching is not exhaustive.

Here is an example of a value that is not matched:

```
_ :: []
```

```
val destutter : 'a list -> 'a list = <fun>
```

pattern matching = mecanismo para fazer análise de casos

**todos os casos devem ser considerados!**

o compilador é capaz de detectar que falta um caso, e até indica qual



```
let rec destutter = function
| []           -> []
| x::[]        -> [x] (*igual a x::[]*)
| x :: y :: rest ->
    if x = y then destutter (y :: rest)
    else x :: destutter (y :: rest) ;;
```

```
val destutter : 'a list -> 'a list = <fun>
```

```
destutter [1;1;2;2;2;3;1;4;2;2];;
```

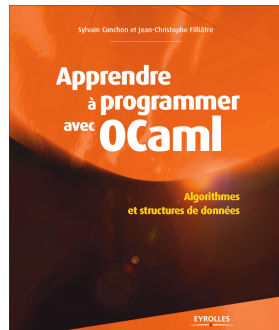
```
- : int list = [1;2;3;1;4;2]
```

---

Conclusão. Quer saber mais?

As aulas de introdução à programação OCaml apresentadas nesta UC baseam-se em duas fontes essenciais:

- **Apprendre à Programmer avec OCaml** (um *must read!*, embora em francês...).
- Sebenta **Introdução à Programação Funcional em OCaml** de Mário Pereira e Simão Melo de Sousa (link)



Adicionalmente ou alternativamente, as referências seguintes introduzem OCaml de forma completa:

- **Real World OCaml**
- curso online: Introduction to Functional Programming in OCaml ([link](#)) (esta aula de introdução é um espelho da aula 0 deste curso)
- **Developing Applications with Objective Caml** (pdf/html online [aqui](#))

