OCaml Users Survey 2020
Summary of results

OCaml Software Foundation

November 22, 2020

1 Introduction

The idea to conduct a yearly survey on OCaml usage was put forward at the 2019 meeting of the Advisory Board (AB) of the OCaml Software Foundation (OCSF). Muriel Shan Sei Fan and her colleagues from OCamlPro provided a first draft of the survey, which was then tested and updated by the Executive Board (EB) of the OCSF, then put online via Google Forms by Kim Nguyen from the EB. It was announced via the discuss.ocaml.org forum and a number of mailing lists.

The survey ran from October 17th 2020 to October 30th 2020. It attracted 745 replies. The Google Forms summary of results is available at `https://docs.google.com/forms/d/1OZV7WCprDnouU-rIEuw-1lDTeXrH_naVlJ77ziXQJfg/viewanalytics`. An alternate presentation and analysis of the results is available at `https://patricoferris.github.io/ocaml-survey-analysis/`.

In section 2 of this document, we summarize the results and briefly comment on each question. Section 3 draws preliminary conclusions.

2 Survey results

Conventions In questions that offer agree/strongly agree/disagree/strongly disagree replies, we aggregate agree/strongly agree replies and disagree/strongly disagree replies, yet put the “strongly” numbers in parentheses. For example, “40% Agree (10% strongly)” means that 40% of the replies were “agree” or “strongly agree”, and 10% were “strongly agree”.

2.1 OCAML USAGE

For how long have you been using OCaml?

3.5%	you are not using OCaml
20.0%	less than 1 year
36.8%	2-5 years
17.3%	5-10 years
22.4%	more than 10 years

Note: a transcription error left out users with 1 to 2 years of OCaml practice. We hope these users selected the nearest appropriate reply.

How do you rate your OCaml proficiency?

14.9%	beginner
34.6%	intermediate
35.5%	advanced
14.9%	expert

There is a good balance of expertise among the respondents: from one to 10+ years of practice, from beginners to experts.

There was a concern that having so many "old timers" (10+ years of practice) and advanced/expert users among the respondents could bias the results of the survey away from the needs of beginners. Preliminary analysis suggests that the results to the other questions are relatively consistent among the experience groups.

Where do you use OCaml?

10.2%	At school
52.2%	At home
56.7%	At work, in industry
26.7%	At work, in a research job

More than half of the respondents use OCaml in industry as part of their employment. It's not just for teaching and for hobby projects.

OCaml usage at your workplace:

45.5%	Most (> 50%) software projects use OCaml
9.8%	Many (> 20%) software projects use OCaml
17.3%	Some (> 5%) software projects use OCaml
27.4%	I am the only OCaml user

More than half of the respondents (55.3%) work in companies where OCaml is used for many projects (20% or more). It is not just one lone OCaml programmer in the company.

OCaml trend at your workplace:

21.6%	The company is increasing OCaml usage
74.6%	OCaml usage is stable
6.8%	The company is decreasing OCaml usage

OCaml usage at the workplace is overwhelmingly stable or increasing over time.

Which of these other programming languages are you fluent in?

9.6%	F#
21.8%	Haskell
19.1%	Rust
9.3%	Scala
2.9%	Swift
10.9%	Go
55.1%	C
34.9%	C++
41.3%	Java
13.4%	C#
19.4%	Scheme or a LISP dialect
9.8%	Erlang or Elixir
15.9%	Coq, Lean, Agda, or Idris
46.3%	Javascript / Typescript
57.0%	Python
9.2%	PHP
8.6%	Elm / Purescript

The respondents are well versed in “classic” programming languages: Python, C, Javascript, Java, C++. Fluency in other functional programming languages is significantly lower: 22% for Haskell, less than 20% for others.

Which types of software do you develop with OCaml?

35.6%	Data processing applications
13.9%	Numerical programs
23.0%	Formal methods tools
14.1%	Distributed applications (blockchains...)
29.0%	Web frontend
31.0%	Web backend
31.7%	Systems programming
36.1%	Programming language implementations
37.9%	Developer tooling

Other: Analog circuit computation. Analyzer. Astronomy. Audio manipulation. Audio processing applications. Autograding. Automated trading systems. CLI. Combinatorial optimization. Compilers. Courseware. Data structures and algorithms. Desktop GUI. Desktop industrial apps and shared libraries. Desktop terminal applications. Education. Education: sample programs. Educational homework/exercises. Embedded. Financial systems. Games. General purpose libraries. Geometry. Getting started with web. Hardware design. I don’t. Image processing. It’s my go-to calculator and an scripting language. Libraries. Mainly for applying basics of functional programming and applying mathematics. Math riddles. Mobile app. Moving to date processing. Music software. Network protocols. Nothing right now. Parallel processing. Parser generators. Personal scripts/automation. Physical puzzles. Prototypes for research in semantics: expressing meaning of calculi. Science. Scientific modeling and simulations. Security (cryptography). Servers that manage trading risk and regulatory obligations. Simulation. Simulation program. Small pet projects. Small tools. Spreadsheets. Static analysis tool. Statistics. Tiny games. Tools for traders. Trading platforms. Typesetting system. University problem sets. Unix shell scripts. Video games.

The respondents do a lot of Web programming, data processing, and systems programming. The historical applications of OCaml (programming language implementations, formal verification tools, developer tools) are still well represented.

Which industries do you use OCaml in?

28.3%	Banking / finance
3.6%	Commerce / retail
3.1%	Gaming
13.1%	Education
5.2%	Embedded
3.6%	Healthcare / medical
3.1%	Mobile
23.9%	Web
9.9%	Blockchain / cryptocurrency
9.9%	Security
34.1%	Academia / research

Other: Agritech. Audio software. Bioinformatics. Building Automation. Cloud. Computer networks virtualization. Data Storage. Desktop. Desktop apps. Developer tool maker. Developer tooling. Developer tools. Energy. Enterprise software;Factory Automation. General tools for programmers. HR. Hardware development. Hospitality. I don't. Industrial control (PLC) and data collecting (SCADA). Industry. Information Technology. Information science (library systems). Infrastructure. Internal storage system. IoT. Logistics. Machine learning. Networking. Networking appliances. Non Destructive Testing. Nuclear. OCaml service. Object Storage. Object storage. Online publishing. Open source. Personal. Petroleum. Plant logic. Plates manufacturers. Programming language developpement. Public Safety / Public Transport. Publishing. Pulp&paper. SEO. SaaS / dev tools. Scripting. Simulation. Social media. Software consultancy. Software development. Sports. System Simulation. Systems. Telecom. Telecommunications. Television show. Transportation / Logistics. Tubes and bars manufacturers. Typesetting. University library.

It was a mistake to group "research" with "academia", as significant research is also conducted outside academia. At any rate, the financial industry and the Web industry are well represented among the respondents, and so are blockchains and cryptocurrencies.

What is a pain point when learning the OCaml language?

Free-form question. 470 replies. See appendix A.1.

2.2 THE OCAML SOFTWARE FOUNDATION

Did you know about the OCaml Software Foundation (OCSF) before this survey?

51.6%	Yes
48.4%	No

About half of the respondents discovered the existence of the OCSF with this survey. This is excellent advertising for the OCSF!

What topics or actions would you like the OCSF to work on?

72.1%	Support the development and evolution of the OCaml language, compiler and core tools
75.6%	Support the development and evolution of the wider OCaml library and tooling ecosystem
32.1%	Animate the OCaml community (conferences & events organization and sponsorship)
32.6%	Encourage & support OCaml courses for students (Universities, schools etc.)
19.4%	Encourage & support OCaml training for professional developers
22.9%	Increase the diversity of the OCaml community
40.4%	Promote the language in the industrial world (through conferences and events sponsorship)
23.1%	Fund better support across OSes and architectures

Other: Better documentation through better writing and examples!!!. Better documentation!. By far: better documentation. Much, much better documentation. :(. Coordinate efforts to reverse balkanisation of ecosystem into Lwt vs Async, Core vs Containers vs Batteries, etc. Create more learning material and lower the barriers to entry. Take python or php documentations as an example. Dev environment. Documentation! Documentation!! Documentation!!!. Don't bring any diversity bullshit on agenda. Encourage cooperation between companies. Everything or nothing. I expect the OCSF to work on all of the above. Improve documentation and (interactive) learning material. Invest in improving performance of generated code. Invest into modern marketing and community-building efforts. Invite good technical articles (success stories) of OCaml from experienced developers (copy-edited by experts), both intermediate and advanced level, on a bi-weekly basis. where both newcomers can jump into, and OCaml programmers can improve their habits. it may be as easy as "how to read a file", looking at exceptions and error results, over "how to configure your service on both IPv4 and IPv6", or "how to trace and improve performance of a real application". it may need work to convince the experts, and money to pay for copy-editors, but it's worth it. I like jane street's blog very much. Native utf8 for identifiers not just ascii (see whitequark's github). Push for the modernisation of the language. Solve the problem regarding multiple definitions of Map, etc... it makes watching tutorials really difficult when they use different modules designed for the same purpose. Support Documentation of Ocaml Syntax for Bucklescript. Support a larger usage of OCaml through simplification of the entire user experience (installation process, documentation, version features etc.). Support numerical computing initiatives in ocaml such as owl. Unified CLI experience like Elixir, Crystal, Rust, etc. Unite the community, keep in touch like this survey, start projects that are cumbersome but useful like develop support of OCaml for IntelliJ (It is a practical adventage not being forced to switch the IDE when developing C++, Java and OCaml). User experience. We need better architecture of the ecosystem.

Many topics for OCSF actions were supported by the respondents, and even more topics were suggested as free-form input. Among the respondents, priority is given to supporting the development and evolution of the language, the tools, the libraries, and the wider ecosystem (more than 70%+ of the responses). Priority number two seems to be promoting OCaml towards industrial uses and animating the community (30–40% of the responses). Teaching and training comes third (20–30% of the responses).

2.3 COMMUNITY

Where do you interact with the OCaml community?

30.7%	Caml-list
66.4%	Github
23.8%	Conferences (Academia)
6.5%	Conferences (Industrial)
31.3%	Reddit: https://www.reddit.com/r/ocaml/
59.7%	discuss.ocaml.org
14.0%	Slack
23.9%	Twitter
28.9%	IRC / Discord

The discuss.ocaml.org forum, launched in 2017, is very popular (nearly 60%) among the respondents, more so than older media such as the caml-list mailing list or the Reddit group r/ocaml. However, Github is even more popular (66%) as a place to interact with the OCaml community, which indicates that many of these interactions are about collaborative OCaml development projects rather than, say, helping newcomers and sharing news.

2.4 PROJECTS & CONTRIBUTIONS

How many OCaml Projects did you contribute to recently (this year)?

23.0%	None
19.3%	One
33.9%	2-3
17.3%	3-10
6.6%	10+

What constitutes an "OCaml project" is not very well defined. At any rate, the majority of respondents (58%) contributed to at least two projects this year.

What is the size of the biggest OCaml project you contribute to?

25.8%	one-person project
32.5%	2-5 active contributors this year
24.9%	6-20 active contributors this year
8.8%	20-50 active contributors this year
4.2%	50+ active contributors this year
6.5%	100+ active contributors this year

On which platform(s) do you develop OCaml on?

83.1%	Linux
3.7%	BSD
34.2%	Mac OS
11.1%	Windows

Which platform(s) do you target?

93.1%	Linux
12.7%	BSD
40.2%	Mac OS
28.5%	Windows
4.1%	iOS
4.5%	Android

Other: AIX, HP/UX. Any other target if there is no extra effort. Arduino. Arm based Linux. Baremetal. Baremetal systems. Browser. Browsers, via js-of-ocaml. Cloud. Embedded. Embedded (experimental). Generally, I try to be as platform-independent as possible, but I can't easily test on other platforms as I don't have easy access to them. Html. Illumos. JS in browser and node.js. JavaScript. Javascript. Javascript & would love to target wasm. Javascript / browser. Js-of-ocaml. Linux ARM64. MirageOS. MirageOS unikernels. Playing a bit with Ocapic on Arduino. No luck yet. Raspberry. ReScript. Ubuntu touch. Unikernels. Web (JavaScript). Web (js-of-ocaml). Web browser. Web browsers. Web, view bucklescript/rescript.

Linux, macOS and other Unix-like systems are overwhelmingly dominant, both to develop in OCaml and to deploy and run the resulting programs.

Windows is used as a development platform by only 11% of the respondents, but matters as a target platform for 28%. This suggests that cross-compilation for Windows from Linux (or WSL) would be useful already, if not as ambitious as full Windows support in the OCaml ecosystem.

Mobile platforms (iOS, Android) matter for nearly 10% of the respondents, even though their support is currently primitive.

2.5 COMPILER

Which of these language implementations are you actively using?

91.4%	vanilla OCaml
30.8%	OCaml with js-of-ocaml
16.0%	Reason with Bucklescript
9.3%	Reason without Bucklescript

Reason and Bucklescript (and presumably their newer incarnation, ReScript) are used by only 25% of the respondents. Js-of-OCaml is more popular (31%). Only 8% of the respondents are not using "vanilla" OCaml at all. Among beginners, vanilla OCaml is still the most popular choice (77%), but Reason with Bucklescript come second (31%), ahead of Js-of-OCaml. It is hard to say whether this is an accurate picture of the OCaml+Reason world, or a survey bias (maybe the survey was not advertized enough in the Reason community).

Which version of the OCaml compiler do you use most today?

2.6%	4.02
1.6%	4.03
3.6%	4.05
9.1%	4.06
10.8%	4.07
17.4%	4.08
22.1%	4.09
40.3%	4.10
54.7%	4.11
5.1%	trunk

What is the oldest version that you care to support in the software you write?

29.6%	Only the latest release
12.0%	Debian stable
7.5%	The current version supported by Bucklescript
4.6%	4.02
2.5%	4.03
3.6%	4.05
5.3%	4.06
6.1%	4.07
11.1%	4.08
6.5%	4.09
6.2%	4.10
5.0%	4.11

Old (as in 3 years old or more) versions of the OCaml language and compiler are not used much, and their support is not a priority: OCaml versions prior to 4.07 are used by 16% of the respondents only, and deserve supporting for 28% of them. This is a clear message to be sent to the core OCaml developers and the OPAM package maintainers.

Which installation methods do you use?

80.3%	opam (using the public repository)
20.5%	opam (using a private repository)
11.4%	esy
10.3%	npm
13.7%	Distribution's package manager
19.5%	Source
5.9%	Binaries
14.6%	Monorepo

*Other: A custom Debian-based manager. Asdf. Company's repository. Default installation at work. Docker. Duniverse. Internal. Internal dpkg. Nix (I wrote opam2nix). Nix + Nixpkgs. Nix package manager. Nix when I can. Of course, *not* with opam2nix. Nixpkgs. Opam / esy with git. Opam-bin. Would like to start using Nix, also curious about Esy.*

OPAM is overwhelmingly preferred for installing publicly-released OCaml libraries and programs. The use of private OPAM repositories is also getting popular. Other package managers are not used much.

2.6 TOOLING

Which build tools are you actively using?

78.6%	dune
39.1%	makefiles
10.7%	bsb
1.3%	bazel
15.6%	ocamlbuild
4.3%	omake
18.5%	Other (ocp-build, jenga, remake, etc.):

For builds, Dune is very widely used, and traditional makefiles are losing ground. OCamlbuild, once the recommended build tool for simple projects, is disappearing.

Which editors do you use?

49.2%	Emacs
38.0%	Vim
36.2%	vscode
3.5%	Sublime Text
1.5%	Atom

Other: Acme. Acme, neovim (I don't use Vim, but neovim is compatible enough that I ticked Vim above). Codio's education web IDE. Efuns (fork of Fabrice LeFessant Efuns). IntelliJ with the ReasonML plugin. Intellij. JetBrains IDEs (preferred), mainly Intellij IDEA Community Edition. Jetbrains IDEs. Kakoune. Kwrite. Nano. NeoVim. Neovim. Nova.app. OcaIDE, Notepad++. OniVim. OniVim 2. OniVim2. Onivim. Onivim 2. Onivim2. Pycharm. QtCreator. Spacemacs. Spacemacs vim emulation. Spacemacs with evilmode. Students are frustrated with poor REPL access from Atom & vscode. TextMate. Very rarely, VSCode. Webstorm. Wincaml.

Since its inception in 2015, VSCode has become quite popular among OCaml users. Emacs and Vim remain widely used.

Which version control systems are you actively using?

87.9%	Git
1.9%	Subversion
20.9%	Mercurial

Other: Darcs. Fe. Fossil. Hg + internal tooling. Iron (at Jane Street). Pijul.

Unsurprisingly, Git wins. Surprisingly, no one replied “CVS”.

Which tools do you use to test OCaml code?

71.8%	separate unit tests (unit tests in a separate module, as assertions)
31.6%	inline unit tests (unit tests within the tested code, as assertions)
46.2%	expect-style tests (recording the expected test output along with the test)
25.8%	QuickCheck-style property-based tests (random testing)
8.6%	AFL/Crowbar (whitebox fuzzing)

Other: "acceptance tests": proprietary framework in our continuous deployment system for testing in a dev or soft-prod environment. A lot of customs tests too. Ad hoc testing. Alcotest. Cram tests. Cucumber, other ad-hoc integration tests. Custom end-to-end tests. Distributed systems testing + end-to-end-testing. Extensive functional test suite. Functional tests. Have plans to add fuzzing but haven't done so yet. Already doing smoke testing of live services. Home made tool. I do not test yet. Integration tests, run by a shell script. Jest (JS). Mdx. Non-regression tests with our own (OCaml) tool. None. None so far. Not much testing lately. People. Reason Native - Rely. Run with valgrind (OCaml/C integration). Compare OCaml output to C output (wrapper around C library). Selenium. Separate whole-program end-to-end tests. Tests in the module under test. Types > tests. Verification inside Coq. We use inline ppx-test unit tests, plus more complex integration tests in separate modules.

The respondents are well versed in traditional testing methods (unit tests, “expect” tests). Randomized methods (QuickCheck property testing and AFL fuzzing) are not that popular yet.

Which tools do you use to benchmark OCaml code?

79.1%	Whole program time measurements (time, perf-stat, valgrind, gprof...)
32.8%	microbenchmarking library (core-bench, benchmark, bench...)
16.1%	preprocessor-helped microbenchmarking (ppx-bench...)
10.8%	performance-monitoring libraries (landmark...)

Other: Bechamel. Benchexec. Create my own tooling. Don't know how to use them. Have not benchmarked much. High res timer, code paths. Https://github.com/copy/gdbprofiler. I don't benchmark. I find all the tooling for this so bad I almost never use it. Jmeter. Manually checking runtimes. Memprof. My own poor's man profiling library. No need to benchmark yet. None. None so far. None, as these tools do not work well on 4.03 on Windows. None, someone else is in charge of the benchmarking and I don't deal with that part. None, unfortunately. Not an issue in our projects. Not used. Ocp-memprof. Reading the generated assembly is enough to see obvious deficiencies, and sometimes to fix them. Semi-automatically inserted instrumentation. Simple Sys.time-based tools. Simple Unix.gettimeofday. The chrome://tracing event format, json based.

2.7 INFRASTRUCTURE

How do you deploy OCaml applications?

28.6%	Through our users' package managers
27.9%	Through the web
25.6%	As docker images
16.6%	As dynamically-linked binaries
52.9%	As static binaries

Other: *As an unikernel. As client-side only web applications. As opam packages. As source in Github. Compilation from source on a compute cluster. Custom. Custom ISO distribution. Custom company tools. Custom solution. Dedicated script to build from source or complete virtual machine. Esy npm release. Github. I don't. I don't as my project is a library. I don't know yet, I haven't gotten that far. In pre-installed virtual machines. Never did that. Nix. Not handling that part of the projects. Npm. Opam. Opam packages. Opam2nix (yes I use it for prod). Source. Tarball.*

The majority of respondents use static binaries for deployment, which is surprising given that building a statically-linked executable is very hard under Linux and nearly impossible under Windows and macOS. Package managers, Web applications, and Docker images are also widely used.

If you deploy on the web, what platform(s) are you using?

54.7%	Self hosted servers
28.5%	Hosting provider
30.5%	AWS
13.4%	Google Cloud
4.1%	Microsoft Azure

Other: *CI-as-server less-compute. Digital Ocean. DigitalOcean. DigitalOcean droplets. Don't deploy to the web. GitHub. Github. Github Pages. Github pages. Github pages for static website using jsoo as a frontend. Gitlab. Heroku. Kubernetes. Local server for developer tools with a UI. N/A. Netlify. None. OVH. Scaleway. Tencent Cloud.*

2.8 FEELINGS

I feel welcome in the OCaml community

68.7%	Agree (30.9% strongly)
18.2%	Neutral
4.0%	Disagree (1.0% strongly)
9.2%	I don't know

Good feelings towards the OCaml community: only 4% of the respondents perceive it as unwelcoming. Note the high number of neutral and "don't know" replies (27%), however.

I am satisfied with OCaml as a language

85.7%	Agree (30.9% strongly)
9.7%	Neutral
4.2%	Disagree (0.3% strongly)
0.4%	I don't know

High satisfaction with OCaml as a programming language.

OCaml tooling has improved in recent (< 10) years

79.7%	Agree (51.9% strongly)
5.6%	Neutral
2.0%	Disagree (0.4% strongly)
12.7%	I don't know

With a record 52% of "strongly agree" replies, there is a strong feeling that OCaml tooling improved recently.

OCaml tooling provides a comfortable workflow for me

63.3%	Agree (15.2% strongly)
20.9%	Neutral
13.1%	Disagree (2.9% strongly)
2.7%	I don't know

OCaml tooling is confusing to newcomers

62.8%	Agree (22.5% strongly)
21.1%	Neutral
6.1%	Disagree (0.5% strongly)
9.9%	I don't know

The tooling is perceived as comfortable (63% agree, 13% disagree) but also as confusing to newcomers (63% agree, 6% disagree).

I am satisfied with OCaml's build tools

57.0%	Agree (13.5% strongly)
24.1%	Neutral
14.3%	Disagree (3.0% strongly)
4.6%	I don't know

A majority of respondents is satisfied with the build tools, but by a narrow margin (57% agree, 14% disagree). Also note 24% of neutral replies, which can be read as "not really satisfied".

What would you change concerning build tools?

Free-form question. 298 replies. See appendix A.2.

I am satisfied with OCaml's package repositories

57.9%	Agree (14.0% strongly)
20.4%	Neutral
8.9%	Disagree (2.8% strongly)
12.9%	I don't know

A majority of respondents is satisfied with the package repositories, but again it's not a landslide (58% agree, 9% disagree, 20% neutral).

What would you change concerning package repositories?

Free-form question. 163 replies. See appendix A.3.

I can find OCaml libraries for my needs.

46.2%	Agree (6.2% strongly)
27.7%	Neutral
17.2%	Disagree (5.2% strongly)
5.5%	I don't know
3.3%	I don't need libraries

A solid minority of respondents (46%) is satisfied with the libraries, but many are unhappy (17%) or not really satisfied (28% neutral).

What libraries are you missing the most?

Free-form question. 198 replies. See appendix A.4.

OCaml libraries are well documented.

24.6%	Agree (2.6% strongly)
35.5%	Neutral
31.5%	Disagree (8.3% strongly)
8.6%	I don't know

The respondents are definitely unhappy about library documentation: only 25% find it good.

OCaml libraries are stable enough for my needs.

68.8%	Agree (11.7% strongly)
17.8%	Neutral
4.8%	Disagree (0.6% strongly)
8.7%	I don't know

Stability, however, is not an issue: 69% find OCaml libraries stable enough.

I have a good understanding of OCaml best practices

54.3%	Agree (10.9% strongly)
24.0%	Neutral
16.3%	Disagree (2.3% strongly)
5.4%	I don't know

Another question with a short majority (54% agree) and a lot of neutral replies.

Software written in OCaml is easy to maintain

76.6%	Agree (32.8% strongly)
15.6%	Neutral
2.5%	Disagree (0.3% strongly)
5.3%	I don't know

There is consensus (77% agree) that OCaml software is easy to maintain, with 33% of "strongly agree" replies.

As a candidate, I can easily find OCaml jobs

9.6%	Agree (2.0% strongly)
13.6%	Neutral
34.6%	Disagree (15.1% strongly)
42.3%	I don't know

Not all respondents have experience looking for an OCaml jobs, as suggested by the 42% of "don't know" replies and the 14% of neutral replies. For the other respondents, the job search experience is globally negative (35% disagree, 10% agree).

As a hiring manager, I can easily find qualified OCaml candidates

5.6%	Agree (2.0% strongly)
14.4%	Neutral
18.8%	Disagree (6.7% strongly)
61.3%	I don't know

Likewise, there is a lot of "don't know" and neutral replies, suggesting that not many of the respondents have had experience hiring OCaml developers. Again, for those who express an opinion, it is globally negative (19% disagree, 6% agree).

2.9 BURNING DESIRES

If I was granted one new language feature today, I would ask for:

35.2%	a multicore runtime
19.8%	typeclasses/implicits
11.3%	a builtin "deriving" mechanism
15.4%	an effect system (a type system for side-effects)
3.8%	effect handlers (user-definable side-effects)
3.7%	a more powerful module system
4.0%	a complete redesign of the surface syntax
2.1%	namespaces
4.8%	no new feature, please!

This was a tough question because only one choice was allowed, and the order in which the choices were listed may have influenced the replies. In retrospect, multiple choices presented in a randomized order would have been better. At any rate, the most desired features are the multicore runtime system, type classes / implicits, and an effect system. (This ranking is the same for beginners, intermediate and advanced respondents; experts give priority to type classes / implicits.) A complete redesign of the surface syntax is not desired, even by beginners (10% vs. 4% overall). Namespaces, modules, and effect handlers attract little interest.

If one piece of the ecosystem could magically be made state-of-the-art, I would ask for:

14.9%	a package manager
15.6%	a build tool
6.2%	a language website
13.4%	a compiler
5.9%	a support community
8.9%	documentation for the core language
18.0%	documentation for user libraries
10.8%	communication to outsiders (blog posts, conferences...)
6.2%	my domain-specific library ecosystem

Here as well, multiple choices could have helped identify trends. The actual results are all over the place, with no single piece of the ecosystem attracting more than 18% of the replies. Documentation for user libraries, build tool, and package manager come first, echoing the dissatisfaction or mild satisfaction with these aspects of the ecosystem expressed in earlier questions.

What do you think are the main pain points that prevent OCaml adoption for new projects? (Pick at most three)

50.4%	Too hard to find and hire OCaml developers
29.4%	Too hard to learn
15.9%	OCaml does not fit your environment constraints
8.0%	Lack of performance
51.1%	Lack of critical libraries
38.7%	Lack of critical developer tools
3.6%	Lack of backward compatibility

Libraries and hiring difficulties are perceived as the main obstacles here; both are mentioned by a majority of respondents. Tools and learning difficulties come next. Performance and backward compatibility are generally not an issue.

Tell us anything!

Free-form question. 229 replies. See appendix A.5.

How should we improve this survey in the future?

Free-form question. 95 replies. See appendix A.6.

3 Preliminary conclusions

3.1 OCSF actions

The executive board of the OCSF is glad to see so many replies to this survey and is receptive to the priorities and suggestions expressed in the replies.

The OCSF already funds the development and maintenance of several libraries and tools, and has plans to fund more projects of this kind. It takes good note that library documentation is often lacking and should be an integral part of library development projects.

The OCSF is also planning to increase its support for the evolution of the OCaml language and core system, and to support technical writing actions to improve tutorials, reference documentation, and the ocaml.org website. These planned actions are limited by the current OCSF budget. To go significantly further will require an increase of sponsor donations.

3.2 Other impact

The survey sends some messages to the core development team in charge of evolutions of the OCaml language and of the compilers, such as: focus more on new language features (implicits, effects) and less on wringing the last ounce of performance and maintaining compatibility with old versions.

Another message is being sent to OCaml Labs: the “OCaml Platform” as initially envisioned, comprising a consistent set of curated, consistent, and well-documented libraries, is as relevant as ever.

A more general message is addressed to all the developers of OCaml libraries: writing these libraries and packaging them in OPAM is good, but documenting the API is a necessity. Tools could be considered to help send this message, such as a “linter” that analyzes (the lack of) documentation comments in OPAM library packages.

3.3 Future editions of this survey

It seems worthwhile to conduct a similar survey every year. Several improvements should be considered. First, the list of questions should be adapted (some added, some removed) based on the feedback we got this year. Second, free-form questions should be avoided or their use reduced, as we are unable to exploit the mass of replies they produce. Third, good survey practices such as randomizing the order of replies should be followed. Fourth, it would be ideal to delegate the formatting of the survey and the summarization of results to professionals.

A Replies to free-form questions

A.1 What is a pain point when learning the OCaml language?

Lack of over-loaded numerical operators (though have mixed views on ad hoc polymorphism in general)

Double semicolon in examples, no main function

Incoherent tooling (hard to grasp opam+dune+...), missing features out of the box (no derivers, no printers, no JSON, ...)

Classes

Graphical User Interfaces

Documentation

Perhaps the amount of "folk knowledge" around coding styles and what libraries are considered "good" to use. Of course this is very hard to do especially in a non controversial way. Different people have different needs/perceptions, etc...

Module language vs. standard language

Too many tools (opam + dune), lack of easy support for cross compilation, debugging/profiling tools need more work to get easier to use and need better advertising and documentation !

Examples on boilerplate for projects. Directory structure, how write tests. Those are mainly found in random blog posts on the internet

the build system

Students around me are used to fancy IDEs and C-like languages, so running something form the terminal and sometimes even GNU/Linux scares them off

Lack of good documentation beyond simply learning the language

OPAM

Lack of example code for many libraries

like any programming language - about two weeks in

no decent (free) documentation (as far as I know the manual is still incomplete), inergonomic and error prone syntax (bracket-less)

Missing commonly agreed upon standard library; ocamlbuild missing parallel builds; runtime support for concurrency

When learning: unfamiliar syntax, large number of paradigms/systems (functions, modules, functors, objects, etc.). When using in general: lack of good support for ad-hoc polymorphism e.g. type classes/traits/interfaces (first-class modules are a little awkward to use)

Syntax errors

Syntax error reporting

The use of the let keyword for toplevel bindings, val would be better. Weird priorities with if statements (depending on whether it starts with a let).

Sparse documentation

IDE support

Getting started (installing a specific version and obvious packages) is a bit more painful than Python.

Tooling; documentation

No Windows Plattform

Bad error messages, no type level interpreter

Libraries feel incomplete, or missing

the 25000 different syntaxes tooling expects (.merlin, .ocamlformat, dune, opam) next to the already new syntax of OCaml.

Old libraries that throw exceptions

Fragmentation between the docs that use Core, and the ones with the standard library.

; vs ;;

ecosystem

Too many features, somewhat messy syntax

compilation without dune / ocamlfind

lack of documentation

1. Setup! i.e., installing all of OCaml, opam, dune, editor support, course libraries for MacOS and Windows (WSL Ubuntu), it's a huge barrier to entry. Setup for students should be much simpler. 2. If it was easy to write web apps in OCaml, many, many more students would be attracted to it.

The lack of a "pret a coder" platform that we could just install and run code with sane tools and defaults

Weak polymorphism

The type system is strict, which is a learning curve. But you learn to love it!

;;

Documentation or other materials explaining the benefits and drawbacks of accomplishing "the same thing" in the language. For instance object-based library interfaces versus module-based ones, or exceptions versus options, etc.

In a classroom, it's quite annoying to have all the students with their laptop dully configured with "the OCaml platform"

The lack of a Base-free version of "real world ocaml"

Syntax

no namespacing, bad recursive modules, no ad-hoc polymorphism

Lack of high level frameworks

Number of tools to setup

Understanding best practices for things like testing. Not enough hands on tutorials building non trivial programs.

Unhelpful syntax error messages, poor standard library, no up-to-date and comprehensive tutorials (ocaml.org is out-of-date, there is no comprehensive tutorial for dune, ...)

Lack of standard resources. The manual is terse and assumes a lot of prior knowledge. The Cornell course is dated, not kept up-to-date with the language developments. RealWorldOCaml reads like an on-boarding manual for new-hires at Janestreet with all the unnecessary complications of the arbitrary stylistic choices made at their company. Useful information is scattered in blogs and fora.

API docs aren't as pervasive and easy to use (and search!) as they could be.

Up to date docs and learning materials and books.

Lack of search engine friendly content. Other mainstream languages have plenty of short tutorials available on the web. The few OCaml does have are not discoverable easily

Dune

Documentation

Language is too restrictive coming from Haskell.

At the time, it was the lack of documentation, though this was almost 20 years ago, so I don't think my comments are particularly relevant to the current situation.

Syntax. ReasonML helps. Hard to find sample code. Library ecosystem is rough. Have to dig into library code to get them working sometimes

Lack of narrative documentation (tutorials, large-system design guidelines like found in the object-oriented world)

For me, no real pain point ; for others (like students I would teach OCaml) may be the unusual syntax and semantic compared to C or Fortran (that is missing in the list of language my community use). But I do not think the Reason (C/Java like) syntax is good for the community. Better if new user with the help of tutorial try to wrap their mind on the syntax.

Object system

Less guided tutorials, both written, video, and even podcasts.

Tooling and lack of good examples/documentation

lwt vs async

Not enough tutorials and books on advanced topics (functors, GADTs, polymorphic variants, etc)

The barebones Stdlib and consequent need to use a 3rd party replacement.

Using it on Windows.

Learning how to read the codefollow its structure. (So much better than other Lang's in the end!) Relation and difference between library names and open packages. Trying to understand some compiler errors. e.g. Trying to define negative floats with -. Or the when the func types are long it's hard to distinguish where is the mismatch.

Fragmented ecosystem, incompatible standard libraries, lack of libraries

- poor tooling: - IDE - there are several problems here: - Small support for OCaml support in modern IDEs such as VS Code. There is effort, but it's mainly by enthusiasts who spend their free time on the extension and ocaml-lsp development rather than it being a more stable development effort. - In general, merlin is far behind in language support in comparison to other languages' editor support. An example would be that there is no "rename all occurrences of an identifier". - debugger: having no modern debugger feels awful in comparison to debuggers in JS, Rust, Scala, Go, any other popular language. - There are very few resources to learn how to develop with OCaml besides RWO and OCaml Manual. I wish we had a strong community driven OCaml version of The Rust Book, which explains complex ideas in a very down-to-earth way that even a beginner feels confident with advanced Rust language features (this isn't the case with RWO or OCaml manual)

Lwt, async divide

Standard Lib Vs containers Vs core Vs base etc

Tooling

Understanding how the various systems interact, so you know how to fix compiler issues about version mismatches, or ppx issues.

None

where do I begin.. - there are no beginner tutorials, making it incredibly time consuming to get started to the point where most people just quit. I found it easier to get started with Rust for example, which has an amazing documentation and detailed beginner tutorials. - the libraries documentation is very lacking both real explanations, but also examples; for example compare a typical OCaml https://ocaml.janestreet.com/ocaml-core/latest/doc/base/Base/List/index.html and F# documentation: https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html - there are no "quick get going" tutorials on how to setup all necessary tools to get started (opam, dune, common workflows and how people setup their projects) - the concept of opam and that there are no lock files by default - other package managers like npm and paket make it so easy to setup consistent builds, environments and CI; with opam I still don't know how to setup a performant CI that doesn't take 40 minutes to download packages - there are no suggestions on patterns and code style, F# has pretty good ones: https://docs.microsoft.com/en-us/dotnet/fsharp/style-guide/ - to be honest if it weren't for F#'s documentation and ReasonML's beginner tutorials I would have quit on OCaml a long time ago - the OCaml ecosystem is kind of doing its own thing in a bubble making something different than everyone else, making it difficult to port skills. A good example is atd gen - instead of having a good grpc implementation to make it easily possible to interact with other systems and languages, the most common serialization way is incompatible with anything outside of OCaml.

Documentation, setup of compiler/package manager/tooling (when developing at home)

modules are unintuitive, especially when contrasted with Haskell-style typeclasses

Too many choices for coding style. Becomes a blocker for onboarding. Split community on async/lwt. Adds to mental overhead. Where to put comments? Before, after, in a different file? Writing interfaces in different files. Ecosystem small for data processing (file formats)/protocols. Protobuf, grpc, openapi. Database libraries. We have 4-5 for postgresql. Yet only one supports ssl. Profiling, tracing and instrumentation. The above is unstructured. And a brain dump. I find ocaml great though :)

The fact that Base is not in the main language makes finding documentation rather confusing.

Lack of quality documentation, very dificult as a new-comer to understand how everything fits together.

It used to be tooling that's hard to put together. Now it's probably lack of resources, really poor official guidance. ocaml.org is beyond terrible.

Access to good standard libraries and code resources/examples

Compiler doesn't warn about inconsistent indentation, making syntax errors hard to find.

Tooling fragmentation. I'm very thankful for dune.

No pain points. I was able to easily pick up the language. Tutorials on the various build systems(dune, ocamlbuild) and testing framework (ounit, alcotest) would be useful.

Setup is difficult. I make an attempt to set up OCaml at home every year or two, and inevitably give up after filing multiple issues. And I have years of full-time professional experience using OCaml to build dev tools to write OCaml. I (maybe inappropriately) mentioned my frustration at the end of this latest issue: https://github.com/ocaml/opam/issues/4377

At work, you start with a full-featured environment with merlin and an incremental build system and emacs integration all set up. Trying to set up OCaml outside of work was so difficult I gave up. (The most recent platform I tried was Ubuntu on WSL.)

(un)stability of the ecosystem, first and foremost of ppx.

Nothing since we have Dune. Before the tool system mess!

The ReScript ecosystem being new and sometimes second class, no batteries-included web server framework option

Function application syntax. "Let in" rather than rust style block of assignments with final expression. Difficulty in getting a working dev environment. Difficulty in running tests (dune support for ppx_test gives you no non-hacky way of running specific tests!). Confusion about which modules relevant values are in (for people who expect the convenience of "methods" where you don't need to prefix with a module name). Basically I think rust made a lot of good decisions with syntax and tooling that we should copy.

There is no cookbook style collection of examples for practical situations.

Learning material / ecosystem.

Different treatment of variables, like any functional language

The tooling back then

Lack of community libraries, and the split between jane street libraries and non-js libraries. Any editor experience without go-to-definition.

Cryptic error messages

Remembering which are the negations of = and ==

The separate term/module/signature/interface languages

Packages management (before esy) was horrific

I'm not sure I recall major pain points in learning OCaml as a language, just that it took some time.

Documentation, video courses

small community

The module language

Finding information on profiling.

Lack of centralized documentation (ala docs.rs)

Syntax familiarity. It's really hard to get many users to even consider the language because the surface syntax seems so unfamiliar.

The community

Error messages are impenetrable

Write idiomatic code

Setting up the OCaml environment requires environment variables e.g. through 'opam env'. This leads to confusion as some shells have the correct environment while others don't and the difference is hard to spot. Applications like editors launched from the GUI don't have the environment variables set causing hard to understand failures. Compare this with go – now that it has abandoned the GOPATH – you can git clone; go build and packages are downloaded and cached and your program is immediately built.

Honestly don't remember but navigating OPAM and dune is tough at first

Syntax for new users. Lack of quality Stdlib.

Confusing type errors

the document is hard to read somethimes

The absence of books (except "Apprendre à programmer avec OCaml") and courses in French.

Lack of curated documentation resources. Syntax looks complex to beginners (syntax errors are a nightmare to debug for beginners). Libraries are designed for advanced users and lack good UX and examples.

Dune (sexp config)

Decentralized ecosystem

It is rhe first language i learnt. No pain

The manual examples have greatly improved, but they are still sometimes terse or could be more elaborate for beginners (I myself have not had problems, but I know the language well.) Take the GADT example for instance - I sent someone who was trying to learn GADTs to that page and they were very confused about the exhaustiveness check example.

No ad-hoc polymorphism (implicit modulars), the need for ppx rewriters such as show and enum

Lack of documentation written in a very condensed format. Sometimes I spend hours reading through dune or some ocaml library docs because they are complete and thorough but fail to communicate how to solve the problems that I have. You could argue that StackOverflow would be better suited for this but the reality is that we aren't that many over there either, so I more often than not end up reading sources

Learning functional programming when it's not where you're coming from

The quality of the libraries

the sometimes cryptic error messages

type system (it takes a while to understand, and its *NOT* a problem of ocaml)

Lack of bindings/interop with foreign libraries

Information integration in the community is not in place. Example: the use and development of ocaml-lsp, .merlin Missing, which made me a headache in the first semester

1. Understanding the difference between standard OCaml and the Jane Street extensions, why people prefer one over the other, and how incongruences between the two can complicated the learning process. 2. Leaving to program tacitly in a category theoretic style (e.g., let%map) without any of the syntactic support I've grown to love in other languages (e.g., simple ad hoc type class definitions, type-preserving macros).

Feature-poor standard library (Base/Core got me over this hump).

Lack of resources, as in tutorials, blogs, books and video courses.

Almost no jobs await for a beginner after hard learning.

Poor documentation

Error messages, terrible tooling

Figuring out how the type errors relate to what's happening in my code

multi core

Reading docs and using libraries. It can be hard as a beginner to use libraries like core without yet having a firm grasp on functors and the edges of the related syntax.

Avoiding overly complex or fragile solutions when getting started and just wanting to get things done. Experimental or unproven approaches should be marked as such, somehow (showing usage trends over the last 5 years could be useful, maybe idk)

Understanding the type system

Diversity of PPX options for deriving code from type declarations.

Lack of good documentation illustrated by examples for programming libraries

Lack of presence when googling for basic questions

Lack of modern tooling

The use of preprocessors makes the code less transparent, and it takes some time to learn how to work with the compile errors.

Packaging and extensive libraries

Proviso: I have not used OCaml much in the last year, so some of the problems I mentioned below may have already been ameliorated. (1) Complex collections such as maps with confusing requirements and interfaces. I know that these have to somewhat complicated because of OCaml's static typing, but I wonder if there's a better way. I just try to stay away from these collections. (2) Documentation: Many library functions lack documentation other than the signature. (3) Documentation: It's been possible to build documentation that will be easily available offline, and in particular available in utop, but getting that set up can be a pain. I've given up on it for the most part. Contrast this with Common Lisp or Clojure, where function descriptions are almost always available with a single command in the REPL by default: the programmer doesn't have to configure anything to make this available, and there are no functions where the documentation build fails–there's nothing to build, since once you have the function, you also have its docstring. Of course, there also has to be a standard convention of always providing documentation strings before functions are made public. (4) Documentation: A standard, central documentation site with user-contributed examples like Clojure's clojuredocs.org would be very valuable. In clojuredocs, for any given function that's included in the database, there are four parts of a doc page: (a) the official documentation string, which can't be edited by most people; (b) user-contributed examples; (c) user-contributed comments; (d) user-contributed "see also" references to other functions. With OCaml, at present, any contribution of examples or other comments on function documentation has to be done at dispersed sites and usually has to be done with a PR or some other slow-moving, carefully vetted process. That level of vetting isn't always necessary, as clojuredocs shows. It takes very little to sign up to add remarks and examples on clojuredocs. I'm sure that there's someone who will delete spam and junk, because I've never seen any on clojuredocs, but I don't believe there's a lot of spam and junk submitted, so filtering it out would be easy. Unlike Java or Python, but like Clojure, OCaml has a small enough community that there's little danger of such a site being overrun by silly or malicious contributions. (5) Jane street libraries are one standard and the most extensive, coordinated set of libraries, but the online documentation is kind of a mess because of different, incompatible library versions, missing documentation of functions, and confusing, multi-layered programming interfaces. One can't really avoid using Jane Street libraries at some point, and given that RWO is a de facto standard introduction to the language, JS libraries could be the de facto standard if it weren't for such problems. The authors of RWO have done a great service to the community, but once someone gets beyond the book, they will at some point find their way to a region of documentation wilderness. (As a result I prefer other libraries when I can avoid JS. Also because function-last map-style functions are contrary to nature.) So those are my pain points, since you asked, but of course I think OCaml is a great language. That's why I use it.

It seems like I am never coding in straight OCaml. There is always some plugin somewhere that alters what something means. It's almost ironic, because the language itself is so straight forward (like functions always only taking one specific type, hence ‘+‘ and ‘+.‘), but then someone loaded something somewhere and then what you thought you knew, no longer holds.

The lack of step by step debugging tool make the learning process really hard

Document and sample code for OCaml libraries

Windows support

it’s syntax, in particular but not limited to: ‘if x then p1 else p2; p3‘ always executing ‘p3‘. also its standard library.

Library documentation

small stdlib

small community, very few business opportunities

Error messages when compiling code with a syntax error. In particular, more so than in other languages, OCaml is susceptible to "run-on parsing" where the compiler-reported location of the error is very far after the actual location of the programmer’s mistake. (I assume part of the problem is the fact that MLs have many "unclosed" syntactical structures like "let ... in ...", whereas a C-like languages would have "let ... in {...}")

The beginner experience on Windows isn’t very smooth compared to Mac and Linux.

When I was learning, the build-tool options, but now dune and opam are pretty slick I think even from a beginner perspective. I’m unsure what a current pain-point might be.

Lack of centralized documentation

Modules

strange syntactic restrictions, boilerplate generation

understanding the importance of types

The documentation is terrible.

Little fresh literature. Small community

Semicolon/conditional precedence. Non-overloaded arithmetic. Functors.

The Ocaml standard manual is strangely ordered and information scattered across section (esp. the extensions chapter is confusing).

Contrary to other language variables are immutable unless mentioned otherwise. It took me a couple month to completely get rid of this habit.

Build system, FFI

Not great editor/compiler set up

No holistic documentation from opam to dune (like rust)

Hard to find practical tutorials about advanced concepts like GADT for instance

Syntax and community

lack of refactoring tools

15 years ago - pretty much everything. Today, things are in much better shape. I would personally focus on an out of the box experience with ocamlformat + merlin + modern editor (vscode, atom) to make the beginner experience as easy as possible.

Modules.

Setting up opam is not well documented at all. On the opam.ocaml.org site, using switches is a minor paragraph - and it makes no sense unless you already know how the tool works. There needs to be a lot of work on docs explaining idiomatic usage, and a examples of usage compared to other language's toolchains. The standard ocamldoc formatting makes it very hard to navigate api docs - there could be a sidebar or something with an index of modules, kind of like how rescript's site does it (e.g. https://rescript-lang.org/docs/manual/latest/api/belt/list)

Non-native repl/utop! (also: for scripts: binary size is unattractive but not really a problem)

Function composition can be annoying.

Some of the syntax errors (like accidentally using = instead of :) produce hard to decipher error messages and are frustrating until you gain intuition for noticing the conditions that lead to those messages

Not an issue for me, but while tooling setup is decent, I think it could benefit from Racket-style distribution, where you get ocaml, opam, common libs, and an already configured basic IDE in one package. Less friction for new users is always better, they're more likely to abandon a language if problems arise.

The documentation. It's fine for people who are experienced in OCaml, but coming in can be very difficult and is a huge deterrent from learning an amazing language.

Lack of modules documentation

The now much solved build management system. We have dune and opam now

Dev tooling

It was mostly pretty seamless for me; the only things I found a bit confusing were semicolons and local let bindings.

Multiple standard libraries, multiple asynchronous libraries

There's no fair answer to this. I find GADT painful. Type inference is great but may be abused and made painful. As a very beginner in 2005, the unit type was a pain point. Haha!

Error messages and tooling

scarcity of modern documentation

dune is quite opaque, without a good manual

Can be difficult to understand how to use some libraries.

Playing sounds

Web development ecosystem

Error messages, 6 years ago, lack of good resources to learn the language by myself

Conventions are different and the ecosystem is on the smaller side of what I'm used to

library ecosystem is poor

Lack of proper usable official documentation. Tooling is a mess (installation, build and testing...)

judging which libbraries are actively maintained. a good non-jane street flavored book. momentum of scientific ocaml overall

Setting up the environment properly

I did not remember experiencing pain points.

Coming from Python & F#, lots of pieces are not available from the stdlib (or even Core) which adds friction when learning e.g. easy HTTP requests, regex, collections, string interpolation, list comprehensions, stdin.

Lack of resources/libraries/documentation

Didn't know where to start with build tools.didn't know how to quickly get some JSON parsing / http request library included.

I'm not sure because I've learned it a long time ago. As for learning new parts of the language, the organisation of the official manual doesn't make a lot of sense for the end user and seems to be organised to match internal development. E.g., whether something is a core language or extended language feature is not something I care about when looking things up.

Syntax, in particular ';;' and precedence with things like multi-statement if branches / match bodies. Also no toString

Find the appropriate tooling. For instance, ocaml.org has tutorials for ocamlbuild and oasis, while dune is the most common. Similarly, many of the OCaml VS Code extensions are deprecated (the OCaml platform one is lovely, but doesn't show up a the top of searches yet).

Frama-C and Infer are developed in OCaml

It's not pure, weak references and impure applicative functors are hard to reason about.

Understanding the module system

I can't recall. That was a long time ago

I learn ocaml after c/java/python and I had a hard time to learn about immutability.

Several build systems, several standard libraries

Not enough current books.

The lack of a good up-to-date (including tooling) tutorial for beginners

?

Completely alien syntax, no magic IDE that does everything

The choice between standard library replacements (e.g. Core, Batteries, Containers). It's very difficult to make a choice without having enough context, which happens when you are a beginnger.

Syntax inconsistencies compared to Haskell

The package systems is bit obscure

Documentation

As a former JavaScript developer, the hardest part is the disconnection existing between OCaml and BuckleScript (now ReScript), and the poor integration of js_of_ocaml with existing JavaScript libraries.

Small base library, fragmented alternatives (batteries, core, containers)

How to organize larger projects, in particular build automation

Unfriendly syntax, ReasonML helps a lot, but still learning how to at least read OCaml is mandatory. Ecosystem is split across several stdlibs/async libraries, adds friction when learning stuff, as one needs to choose what exactly to learn.

multicore+shared mem (but thats coming). modular implicits would be nice to have a more in the language metaprogramming instead of ppx, first class modules can be a bit clunky too

Weak documentation–hard to find information, have to experiment too much

Getting distracted by the competing standard libraries

Build system setup - One click set up projects

Poor library docs

Build system setup for larger projects; docs for jane street libraries

tooling

Lack of GUI IDE - there are multiple pieces that do not always work together.

Difficulty in finding well-used and well-maintained libraries

I don't remember, to be honest.

it is lonely

Poor documentation of community libraries

No multi thread support

None

Lack of examples and lack of more verbose documentation. Reading function signatures doesn't work the best for me, I learn better by examples.

utf-8

No good resources on learning the language

When I learned privately first, build systems were not great. This has improved a lot since.

Error messages around polymorphic variants. I use them a lot with the result type + monad and the error messages are not great if you fail to match on all variants, or return a wrong variant name by accident.

Not the good approach due to previous experiences with other programing paradigm

syntax

None

Two main points: the status of the stdlibs (and explosions of alternatives: batteries, janestreet base/core-kernel/core, ...) and how ppx works

Documentation is hard to read

No namespaces / nested modules, awful standard library.

It was so long that I learned OCaml that I don't remember. Looking at our students, I don't see any particular difficulties or pain points

Async

The syntax

The lack of libraries, documentation in libraries and standard ways of doing things. Too many ways of build setups with ocaml/opam/dune/esy and how they interact together. Build tooling errors are not easy to understand/debug straight away sometimes.

documentation, examples, poor standard library

Syntax

Lack of typeclasses and monad transformers

Setting up and maintaining a stable dev environment

Documentation is missing examples

Most libraries have very sparse documentation.

Error Messages, from the compiler & the tooling can be better. A lot of time is lost debugging errors because the error message led me astray.   More documentation: Javascript/Python style documentation of APIs along with examples.

Being frustrated by lack of typeclasses / modular implicits. The frustration goes away with time

learning curve when coming from a pure OOP culture

Simple tasks not in the standard library (like sending an email) are very difficult to find, one never knows which external package to use, their docs are often limited to reading the .mli. Also, lack of easy-to-use graphical interface to make simple programs more interactive, input text, display plots, etc.

Nothing comes to mind

The lack of and/or strange syntax for polymorphic recursion. Many different ways to format the same code, no standard style. Difficult to write code generation - no real macros even now, and writing ppxs is not easy, so generally don't use code generation unlike in Rust or JS/TS.

Tooling, e.g. debugger not showing abstract types

I spent a lot of time looking up the syntax for things (especially various syntax extensions). It was relatively hard to find the page with the syntax described on it and the BNF doesn't include the extensions. But maybe I'm a weird case in that that was the main thing I was looking up.

Lack of resource, for instance RWO being v2 beta since a while now

Understanding when things are computed, and why there are sometimes computed "too soon".

Format strings (Format4, Format6; understanding when the compiler infers what)

search results on google for simple things are sometimes bad (e.g. direct to a very old version of libraries)

Fragmented ecosystem, hard to get into the tooling.

Bad Stdlib

Lack of documentation / examples online vs other languages

Unhelpful "Syntax error" compiler messages.

Standing up a useful development environment is prohibitively difficult. I wouldn't be productive if my company didn't do most of the legwork for me.

Not really a pain point but a tiny problem I had was using Int/Float as in other languages I was used to casting between the two is easy and the mathematical operators use the same syntax for both (+ not +.)

most pointers online point to old opinionated resources making use of third-party libraries without making it clear that they are indeed third party, and it's easy to assume these third-party libraries are a 'must-have' and fundamental to the OCaml experience.

let and let ... in using the same keyword "let", no end separator for match

Semi colon parsing precedence / bad syntax choices

Documentation. Lack of good tutorial based on async data processing. Lack of tutorial based on creating real web servers.

Confusion about ocamlfind libraries, packages, modules.

it's been a while since I was learning but one pain point was if and match not having explicit end markers, and having opposite conventions for the implicit ones (that is, if statements implicitly end at the first semicolon / "as soon as possible" but match statements last as long as possible); another is lack of type constructor polymorphism, though I understand there are workarounds for that

Lack of unified presentation of tools. They are now great tools (dune, ocamlformat, merlin...) but most ressources does not teach them

Lack of beginner-friendly learning material, I often have to read Haskell materials and translate them. Also it irrationally upsets me that type application is backwards in OCaml):

Inconsistencies between syntax (e.g. "sig :" but "struct =")

Learning the functional programming mindset

Knowing where to find what

IDE and debugging tools

The lack of separating symbols in the syntax; the fact that, absent any parentheses, it is difficult to infer the associativity of operations without being familiar with the language

+.

Documentation of the module system is confusing to me. I consult this page https://caml.inria.fr/pub/docs/manual-ocaml/moduleexamples.html only as a last resort.

the lack of implicit upcasting tripped me up a bunch in the early days, and still represents a 5-minute speedbump occasionally

The syntax around module and type declarations was hard to remember.

I found the core language easy to pick up, but more advanced concepts like GADTs and the ins and outs of polymorphic variants were very challenging to learn.

Sometimes inconsistent syntax

uninformative compiler errors

Lack of good debugger. It really helps to understand a program if you're easily able to pause in the middle and look at all variables.

I feel like it's less Google-able than other languages, both because of the limited size of the community and because we don't use the standard libraries.

Too many language features

This probably isn't my main pain point, but it's what I just thought of: Having several executables for different purposes (ocamlopt, ocamlopt.opt, ocamlc, ocamlc.opt, ...) is really oldschool and kind of strange.

Lack of a turn-key IDE. (This is as of ~4 years ago when I started trying to learn OCaml; it may be better now.)

Bad type errors, Anemic standard library, different OCaml "ecosystems" (JaneStreet vs OCamlPro vs Mirage vs ...), basically impossible to get started on Windows

Standard lib is inconsistent and incomplete. Jane Street's Core is good but not standard. Both Async and Lwt exist.

There's no much information out there, so it's not as easy as "if you don't know it, just Google for it"

Editor support (I use spacemacs). Getting to the documentation of a function sometimes requires several "jump to interface" calls, and I haven't found a way to list all functions in some Module

I miss having a step debugger like in Eclipse. It's also hard to run OCaml locally so I don't practice at all outside of work.

Having to do "module Foo = struct type t = ... end; include Foo; include Functor(Foo)" is annoying

It often seems like Opam packages aren't maintained, or have overly strict dependencies. I feel like I have to include as few external libraries so I have an increased chance of being able to use a standard library from the last year.

the godforsaken state of documentation/online community/active libraries

A lot of the books listed on the ocaml website are quite old, it's hard to know which are still relevant.

Choosing between objects and modules, variants, polymorphic variants and GADT

Creating new projects

The lack of overloading, especially for arithmetics and expressions using modelling libraries.

There are not much online resource and book out there (of course, compared with Java/Python etc but also compared to Haskell). Which leads myself experimenting/reading source code much more frequently instead of relying on conclusion of answers online, which is more time consuming.

poor docs

<rant> Reliance of mathematical ideas without commitment. You'll hear people start to explain type theory, then bail out halfway through with the line "that's the gist". Either commit to a type theory explanation or don't bring it up at all!   One of my favorite lines is from Princeton Companion of Mathematicas: "a well chosen example is sometimes better than a definition". </rant> I don't know if I have a good answer, the existing resources aren't bad.

GADTs

Syntax and a too limited standard library

The syntax takes a little getting used to.

Syntactic ambiguities in the grammar.

Lack of a joined up tutorial, there are many tutorials but they don't join up

Dune

build tools and editor support. OPAM is okay, esy is new and weird. Really would like something canonical that's as easy to use and flexible as PHP's composer

- the weird choices for operator symbols (@@ for apply, but |> for flip apply, and @ for list append, but ^ for string append) - fun vs function naming - semicolons and commas - variants aren't functions (e.g. ‘Some @@ x‘ is an error, even though it makes sense to think of ‘Some‘ as having the type ‘’a -> ’a option‘) - weird uppercase conventions - the Format box algorithm behaves unexpectedly, and its API is counterintuitive/complex and also not good for partial application - weak/insufficient standard library leaves people wanting for more - imperative instead of functional APIs in the standard library

Compared to other modern programming languages, OCaml doesn't come with a industrial-strength standard library. This causes confusion and unnecessary split in the community.

The error messages suck as a new user - I remember feeling like the compiler was always telling me a symptom of the problem, not the problem itself, but I was expecting the problem itself. An example of this is partially applying a function when you meant to supply all of the arguments - often this is not caught until later, where another function could say its mli definition is wrong, when really the mli is right, you just passed a partially applied function instead of the result of fully applying it.

formalizing questions to Google

For absolute beginners: The split between ocaml-stdlib / Base / Core means Stack Overflow answers are somewhat likely not to be immediately relevant. Similarly the split between Async / Lwt.

no debugging capabilities

Documentation of existing libraries, especially standard library replacements. Options for building OCaml are complex (this is less true now with dune taking over the ecosystem though). Poor type errors around some more advanced usages of the type system (GADTs). In particular, internally introduced anonymous type variables are poorly tracked back to a source, causing much confusion when they occur in type errors.

Documentation is sparse sometimes (for example, making the Format module work with auto indentation, you need to know a few more things than what's written in the doc)

older baggage

Functional paradigm

There is a number of unconventional punctuation

Library support for integrations

Getting used to syntax and the functional programming paradigm.

I find the compiler command line really difficult to use directly. To do almost anything interesting I find myself needing an external build system which quickly becomes a barrier to to entry on doing interesting things (even if they're relatively small).

The standard library

Lack of beginner friendly documentation

Windows support, lack of integration between language, standard lib, build tools (until recently - it's greatly improved since I started to learn it)

Setting up a development environment

Lacking documentation and clear examples

Standard library

The syntax (unclosed let and match)

small community, thus small tooling and libs

"(type a)" vs "type a ."

Documentation, comunity (libraries), tooling

The ecosystem has evolved since I learn OCaml (in a good way IMO), so I don't know what's the pain point today

no modern graphic toolkit

Error messages could be bit better sometimes

When I started learning it, back in 2011, there was a clear lack of pedagogical material. It is no longer the case with so many nice books around

Understanding and effectively using the module system. How to use the module system for polymorphism.

Poor editor tooling

tooling is getting better, but still requires tribal knowledge

too little literature past the beginner books/tutorial (many advanced OCaml concepts/features are not documented at all)

Working code from a few years ago stops working :-(

Everything is always incomplete and not working. Multiple "standard" libraries. Bad tooling (build systems, package managment, editor integrations, etc). You can't point someone to a site or book on OCaml to get them started. There will be severe troubles in every single step forward. The OCaml community seems to like killing itself by never finish any tooling/docs before moving on.

New features and syntax; even slightly older books can no longer be used, older code isn't a good reference, e.g. since the elimination of camlp4.

Type Inferencing. Programmers are used to giving the computer commands and when it refuses, they get frustrated and do not understand why.

Small community, so not a lot of online help and you need to search well to find good libraries

online documentation spreading ; bullet proof installation procedure

None, but companies are wary of adding a new language to support.

The most difficult point is the documentation, that is very formal.

limited development environment support, for example for debugging

syntax is awful

It has been such a long time that I can't remember

3rd party libraries

Getting started with build system, tooling. Documentation on things like dune is lacking: appears to presume a lot of predicting background for user, and it's hard to get started using it for non-toy projects. Similar issues with other tools: menhir for example. Great tool: but getting started? No clue how from the docs.

learning FP

Its syntax

syntax

Minor syntax oddities; the lack of documentation for some libraries; lack of publicly available examples for some non-trivial programming patterns;

Documentation / tutorials / examples

nothing

The tools (dune, opam) as difficult to learn. Bad library documentation.

Inconsistent naming conventions for modules, module types and constructors. Docs are really hard to navigate when they leak wrapped module names. Libraries with unwrapped module names or (usually undocumented) optional dependencies are confusing to get started with. Different libraries choose different idioms for hashable types (I think SeededHashType is simple and composes well). The VS Code tooling crashes often and lags when using ppx.

Just the pain points one would expect when learning functional programming, having been used to the bizzare semantics of languages like C/Python/Java/etc, since these are usually the first that one might learn.

The huge difference in code quality between school, industry and academia (from best to worst observed)

How dictionaries work

The syntax is the big one. Not because it's strange, but rather because it constantly requires syntactic differentiation between things which are conceptually the same. As an example, for and while loops contain a block sandwiched between the do ... done keywords, but any multi-line blocks within an if statement must be sandwiched between begin ... end, even though the semantic meaning is identical. Similarly, the inconsistency between curried functions and non-curried data structures means that one can write: f @@ some complex expression, but not Ctor @@ some complex expression, because Ctor technically isn't a function, it's an ADT constructor. But there isn't any difference, in the abstract, between the two: An ADT constructor takes one argument, and returns one value, just like an ML function. This is an abstraction leak: The underlying implementation considers data structures as being different from functions, and so despite the obvious semantic connection, the two are differentiated at the level the user interacts with. The real problem is that all of the weird little interactions between different chunks of the language must be learned by rote, because they do not compose intuitively, coming either from an imperative background or a functional one. This means that in order to start writing OCAML fluently, you need to master a complex and arbitrary corpus of interactions which have no inherent reason for existing (besides, perhaps, backwards compatibility), and have nothing at all to do with solving whatever problem you're trying to use OCAML for. It goes without saying that this is bloody annoying.

"in" nesting vs curly brackets

Tooling: IDE, build systems, package systems. That was more than 10 years ago.

Lack of documentation.

Tutorial, build tool

Initially steeper learning curve

Syntax.

I want more better IDE for OCaml like Scala!

Difficult to understand the many compilation commands

Lack of concise and well-structured textbooks.

Lacking an IDE. Piecing together dev tools, ie dune, merlin, ocamlformat

Its getting better but the tooling

Lack of community developed packages

The lack of documentation for ocaml syntax when working with Bucklescript

running simple IO application, I come from competetive programming background and small codeforces tasks are great way to learn new language and it was super hard in ocaml at the beginning, I was easier when I just filled implementation given at school in a task and running testing script

installing opam

Lack of learning resources, lack of real-world usage examples, limited library ecosystem

Lack of beginner friendly resources

Function/library discoverability: the lack of a hoogle-style web tool to explore the opam repository by type makes it hard to find what you need when learning. This is compounded by lack of function generalisation over higher-arity types, so equivalent functions are duplicated for each type; I think modular implicits/explicits would help here.

I am still learning to "think" functionally.

Understanding advanced usage of the module/functor system

Not having more high quality and reliable source of learning like https://www.fun-mooc.fr/courses/course-v1:parisdiderot+56002+session04/about

Tutorials are hit or miss, and many third-party libraries have poor documentation.

Documentation

It's not too bad, but compared to bigger main stream languages, IDE support is a bit mixed. I think for learners having a 1min setup that provides great auto-completion, navigation, etc. is priceless and allows focusing on what's relevant.

Hard to say as I'm not learning Ocaml at this point. The tooling is way better these days!

Syntax of things like functors can feel pretty confusing.

Understanding best practices and how projects are structured and developed, taking inspiration from some popular open source projects

Too few "easy document". This is maybe bad culture on OCaml community. And many documents target a professional developer. For example, i could not understand about Lwt deeply.

I don't know how to use live-REPL development, where you load the currently-being-edited source file into a REPL and explore it, and I really miss it.

No good learning materials

Lack of good up-to-date documentation

Jane Street's online API documentation has hardly any docstrings

lack of stackoverflow questions & answers, ocamldebug

Authoritative, upstream, up to date, guide; practical patterns; starter guides.

Error messages

The fragmentation of the web stacks : ReasonML, REScript, Ocsigen

Limited size of the community and thus limited forums, books, libraries, etc.

Documentation. Community. Missing/unfit dependencies/packages. Lack of holistic solutions. (Lacks a tried path for basic stuff like webservers and applications. [I know it exist somewhat..] But my stack is still broken - still figuring out things on my own.) I am slower in OCaml world than in any other language I have used so far (JS, Elixir, etc). But I still like the language. I just hasn't been able to get things right yet - I will - hope so. :-)

The ecosystem. We have to build everything ourselves. We even have to build the DB libraries. I've never heard someone tell me "yeah, that'll take 4 weeks to do. 2 weeks to build lib X, 1 week for lib Y and 1 week to actually do the work" until I came to OCaml. It's pretty damning.

Unhelpful error messages from type inference. The fact that .mli files aren't used for type inference and it's frowned-up to put function type annotations in .ml files makes this worse (since even if you annotate the type of a function in the .mli file, that annotation will be ignored for errors inside the .ml file)

The tooling is rough to get used to as a beginner. Setting up a basic project is a chore which usually ends in copying and modifying some other project. Hard documentation is generally good if you can find it, but soft documentation with examples for beginners is often lacking.

Tools in windows

build tooling is still very confusing for a beginner

Modules, functors, 1st-class modules, and all the myriads of syntax that have been gradually hacked onto it over time.

Isolate from other users

Not so many docs

Toplevel let statement and let expression are a bit confusing. OPAM the de facto package manager does not support Windows native environment.

Too academic documentation and weak support for MS Windows OS.

It has a great security level, and it's fast

Lack of ad-hoc polymorphism compared to other functional languages (although first-class modules do alleviate this to some degree).

The biggest one is the utterly awful API documentation – compare with Elixir or Python. Other pain points: OCaml compiler errors could be better. The basic standard library is bare-bones, and there are several competing more-featureful ones bifurcating the community. Windows compatibility is touch-and-go.

Ecosystem

It's been smooth sailing!

Functors

no idea

Figuring out how to use open was difficult

Errors messages remain a persistent sore spot with beginning users.

Realizing when nested matches need parentheses

I came to OCaml from F#, so it was pretty easy for me

Learning which tools and libraries to use

Too many "standard" ways to do things (standard libraries, async implementations, JS compilers, etc). Projects like https://github.com/tmattio/spin are doing a lot to reduce this pain, however.

Parametric types being "backwards" as compared to Haskell. (*) vs (,) for tuples

The tools and how to choose a modern way of workflow with OCaml.

Debugging, or lack thereof.

ppx

installing ocaml and build tools and setting it up

so much syntactic sugar

Documentations. There are just not enough tutorial out there. And most of the the documentations and tutorial are written for other engineers and developers with several years of experiences. We need to write tutorial where a layman can get started. Unless you have a substantive background in coding, most of the explanations will not help.

Find documentation for libraries

Main difficulty when getting into OCaml was learning the process of setting up a project - there's a quite a bit of boilerplate you need to set up before you can build a project, and not many tutorials specify the best practices. Another issue was the lack of a standard library - it wasn't clear which packages I should use, how I should structure the main function etc.

module system is a different language.

Finding the latest and most relevant documentation, selecting the most up to date libs and tooling to use

type shadowing

Finding "ready to adapt" examples

Low quality documentation (type definition only, no description). Fragmented ecosystem.

Documentation on GADTs and the more esoteric parts of the type system.

Windows

Curses, training

Ambiguous syntaxes

Sometimes not so helpful error messages

Low Windows platform support

Understanding when do type, lack of parenthesis and it uniqueness.

syntax

Good resources

Understanding the finer points of the type system

Package Compatibility

A.2 What would you change concerning build tools?

Having a coherent OCaml toolchain distribution.

Improve cross compilation experience

Improve docs

more integration, e.g. Rust's cargo style

tools are complex, evolve rapidly : better explained, small, self-contained examples of different use cases would greatly help!

you need a proper build system and package manager: the compiler cli is more uncomfortable than gcc

I'm dreaming about better support of camlp5 in the merlin/ocaml-lsp

Make tooling more intutive, similar to rust's tooling

Add more examples-based doc to dune and opam

(not sure if this question just refers to build tools or all of the above) a decent debugger is sorely missing, several of the OcamlPro projects (multicore, unboxed data) are very urgent

I'd like better documentation and examples for cross compilation with dune.

Windows support for libraries; ocamlbuild;

Dune: allow an easy way to extend the expect-test + dune promote workflow with the tests and expected input/output written into separate files (e.g. test1.in, test1.expected) - managing dependencies to get this to build and work correctly is quite tricky. Opam: recently asked me to update and when I did, completely broke my Merlin-Vscode IDE integration. (For some reason, all my packages were uninstalled in the update process - ??). I think the overall experience with tools like opam and dune can be very patchy and issue-prone e.g. Dune seg-faults semi-regularly.

Better integration with RPM, i.e. different build and install paths.

Cross-compilation could be easier.

One big problem is that two versions of the same library cannot cohabit. And another problem is that there is no official tooling, only a de facto one.

merge dune, opam, make, etc. config files

Ship the jane street standard library with ocaml?

Improve documentation. More examples and tutorials

Native Windows Support

Strongly deprecate everything but Dune.

my impression is that building small tools (<1kloc) with library dependencies has quite an overhead. dune / custom makefiles are overkill and keeping .merlin / .ocamlinit updated to run scripts via #use in utop is a bit cumbersome as well. a simplified dune setup for such cases would be awesome. but it's not so much of a problem that i'd use a different language instead.

1. I would initiate an Install project along the lines of Opam and Dune. I don't think this has been a priority for the OCaml community because Opam and Dune emerged from meeting the needs of the community, esp. JaneStreet, all of whom already have OCaml installed. (Obviously.) The putative Install project might have a menu of installation options and a single Install button. After clicking it, a person could edit, compile and run OCaml programs on their system. 2. Would it be possible to absorb all of the functionality of ocamlfind/findlib/topfind

into the existing tools? Would it cause some problem to issue a #use "topfind" in an ocaml REPL as part of the boot process for the REPL? I think ocamlfind performs important work, but it seems wrong to have to type: > ocamlfind ocamlc ... 3. As far as the PL goes, it is obviously fantastic! Thank you all! I sometimes lament that variant symbols/constructors and field projections are 2nd class ... I assume making them first class runs afoul of type inference. I sometimes lament the ambiguity of nested match expressions.

Better documentation for opam and dune?

missing up to date .annot for vim + improve scientific support

Have the dependencies tied to the project, instead of being global and having to use switches to change versions of dependencies. (Basically like go module, ./venv in python, or stack in haskell)

Documentation of dune and especially opam is often impenetrable and incomplete. Documentation of commonly recommended libraries is often quite poor.

Improve documentation of build tools

It's horridly difficult to install on Windows.

Better documentation on how all the pieces fit together

Saner defaults (no warning as errors by default), have a real tutorial on ocaml.org, add a dune command similar to ocamlfind for quick testing without having to write dune files, make the devs listen to the community.

The stuffing of dune, ppxlib, opam and other tyranically opinionated and tightly coupled, uncustomizable software down everyone's throats. The 'adoption' is just a network effect and not a testament to the quality of the tooling.

I'd love to have a better workflow for sandboxed builds, where a given project has the source files checked out at whatever versions it needs, and Dune is building the whole thing so go-to-definition works throught the whole codebase. That, plus artifact caching, would be amazing...

Streamlined build for native and web.

I just want to stick to plain old makefiles

Simplify

Ensure reproducible builds

It has gotten better, but the use case of being a distribution manager and trying to build packages from source without necessarily wanting to use opam was not well supported. I'd like this option to remain available.

Source- and binary-level backwards compatibility. Any industrial programming language takes these seriously, and it is frustrating to see how much they are ignored in OCaml development.

Too many build tools, standard libs. Hard to stay up to date

I think there were this idea of a "meta-building tools", a unique tool accepted by the community to do everything (opam+dune+merlin, etc.). Probably a good thing for beginners to have one entry point in the OCaml community.

ocaml-language-server is a fantastic step in the right direction but it's very very slow in large files. Especially ones with any sort of ppx

Windows build of Jane Street libraries

more documentation and tutorials

I would like opam switches to be as fast to create as python's virtual environments

Dune documentation is incomplete and confusing for advanced use cases, in my experience new users have hard time grasping how it works at first. The impossibility to relocate builds is a huge pain

If dune could scaffold new projects that would be wonderful.

Simplify

Adopt a cargo-like tooling, by merging opam and dune

Developer tooling has really improved, but can still be lacking on Windows, especially around paths in spaces etc. Additionally, some new users seem to expect/want a VSCode style debugger. Tooling can also be a bit of a black box, so they don't know to look at dune docs etc, if using esy.

dune has been great, though I find the documentation lacking - it would have been nice to have some common recipes, instead of this: https://dune.build/examples also the documentation similar to other typical OCaml documentations make many assumptions about the users - that they would be already familiar with the terminology and other tools. from the quick start guide: https://dune.readthedocs.io/en/latest/quick-start.html ppxes, byte code, compilation flags, cppo, toplevel - these things don't speak anything to someone just trying out OCaml. they either don't have a place in a quick start or need to be introduced with links to resources where you can read more about

More focus on functional programming ideas

While dune is great, and a big step forward. The number of files becomes annoying, as they gave repetitative content (ppxs). Some kind og project wide dedinitions are fine for inhouse applications that do not care about too many dependencies. So many things to understand in dune because of legacy. It is hard to structure a large project (or folders, modules). In any prog language, but it is far from simple in ocaml. Examples and simple information with trade offs would help.

Tutorials, examples

Library vs package confusion

Better editor integration

I think by "OCaml tooling" you mean the open-source stuff, like dune and opam, which I have not used much. I think our internal stuff is okay, but not great either.

Ease of setting up a full-featured dev environment

focus on making dune a build + package management system (duniverse?), with proper lockfiles (cargo-style, including hash; not just the opam versions which are not immutable). Generally, more opinionated tools like dune. Too many tools is a curse, beginners should only need dune + ocaml-lsp.

Having a support of Mirage and Eliom in dune will be awesome. Having a proper library for SQL queries

add a tool like Rust's cargo as an opinionated, all inclusive default workflow; adopt or create an official language server implementation for better editor tooling

I don't know exactly but I have found cargo is quite easy for beginners to use and dune is confusing

Nothing, dune is just fantastic

Better monorepo support, ability to use "public" libraries in the same repository without its "private" sub-projects also being visible.

The ecosystem got too complex for me. (more documentation?)

Namespaces, esy as default tool, json in dune files, a package.json-inspired or cargo-inspired build system.

The biggest pain point for me (by far) is jumping to definition not working properly. There are a number of open GitHub issues or threads on OCaml discuss about this – specifically that there's a problem where Merlin can't disambiguate between opam libraries that provide the same definitions (https://github.com/ocaml/merlin/issues/894) and a problem where Merlin can't jump to definitions once inside those libraries (https://github.com/ocaml/dune/issues/3276). This directly impedes people's ability to analyze and understand the libraries they want to use. It's also a nuisance that makes the whole OCaml experience feel unpolished.

dune is hard to understand and use in "strange" cases

Better support for nix, better support for bazel. More templates and tutorials for newcomers.

It would be nice to use Dune to easily build documentation for modules internal to my project, with a proper index.

We need opinionated intro material that shows new users how to get started with different kinds of OCaml projects. Defaults matter, so if we can set good examples for new and early users, it will help smooth the process of getting started and reduce a lot of early frustration users may face. Information is also scattered in lots of different locations. A user has to know about opam and dune and ocamlformat and merlin/ocaml-lsp-server and the standard library to get started with a basic development environment. A clear, friendly and centralized intro to the platform would help a lot when introducing new users to the ecosystem.

The community needs a hard reset, to become welcoming to others and alternative views. Even viewing the message boards it becomes incredibly clear that core members are often dismissive and unwelcoming.

Dune has this property where if you don't know exactly how to do the thing that you want, it'll just refuse to do anything and not even tell you that it didn't do anything

I would try to make the ecosystem easier for newcomers. I teach OCaml in some courses: my students told me that sometimes they have difficulties in understanding what tool/library is the best choice, e.g., dune vs ocamlbuild, core vs stdlib vs batteries, etc.

Remove dependencies on environment variables; add "ocaml build" as a subcommand which internally calls "opam" to install dependencies and then "dune" to build and "ocaml test" which runs unit tests. So "git clone" "ocaml build" "ocaml test" "ocaml toplevel" or similar. Tools like "opam" are great but just like "ocamlfind" before it I'd like it to be a behind-the-scenes implementation detail and not something a new user needs to remember and think about. I think "go" does a good job of this (but is bad in lots of other ways, obviously!). Another Go facility worth copying is the "Go playground" where people can run code immediately on the web. Documentation has links to the playground which is really helpful.

Editors (LSP) and build tools.

Centralize the documentation and curate core tools and recommend them on ocaml.org.

Dune sexp config to more modern format

Combine OPAM and Dune

Make the tooling easier and better. Make IDE integration much much better. Add support for things that make ocaml easier to use and write than other languages (use merlin to provide type/value holes, for example) so people get excited about something interesting and different.

Make it easier to publish opam packages, whether its update documentation to include help with cryptic messages and/or a unified build and packaging tool (drum is starting)

improve support for 'other-than-emacs'

PPX

Hope to have more articles to guide the development direction of different stages.I think the configuration of the development environment should be explicitly given on the official website. The use resources I studied are Cornell cs3110 and RWO, I think this is a good way. I hope there can be a better way to integrate learning paths.Sorry, my native language is not English

1. More flexible syntax / better support for DSL development and integration (e.g. defining individual macros without having to create a whole syntax extension). 2. Less magic in the build process - dune relies on conventions that make me uncomfortable in ways that remind me of ruby in the 2000's. 3. Clear guidance from community leadership on navigating the fracture between the vanilla and Jane Street camps.

Barrier to entry is very high with regard to dev environment setup (opam, merlin, dune, ocp-indent, etc.). This could all be encapsulated, or at least very prominently documented for newcomers. Aside: Things improved dramatically with the arrival of dune/jbuilder.

I want cargo for ocaml

Dune book

Dune and opam are honestly great tools, but there are some times where you hit random issues and have no idea how to proceed.

Take some cues from Go. 1. Integrate some external tools into the language. For example, deriving code from type definitions should use an easy, built-in, well-supported mechanism that has nothing to do extending the language syntax. 2. Facilitate application distribution by providing out-of-the-box cross-compilation and static linking. 3. Deprecate or mark as experimental some features of the language or stdlib that result in overly complicated and obscure code, such as various weird extensions of the original module system (first-class modules for example) 4. Namespaces. A namespace is the name of a vendor. A vendor distributes multiple libraries separately. Someone please explain that to Y Minsky and have them put all their stuff into the Janestreet namespace like a good citizen.

Being able to integrate with building other website assets (HTML pages from templates)

I would like to have : A tool to easily refactor code (variable renaming for example), and a well integrated, easy to use, debugger

Make it easier to understand the build process and its many different targets

Integrate all the tools in a Cargo like project manager

Esy helps, just need to mature

If most users can use the same build tools (dune?) then perhaps we can focus efforts and improve performance.

The last time I looked, the Dune docs were very confusing for newcomers. I spent a lot of time in the Dune docs trying to figure out relatively simple things, and eventually had to get answers elsewhere in some cases. That was a year or two ago, so maybe it's a lot better now.

I can't say tbh. I just feel like I have to "ask an adult" to help me get it up and running, every time I need to work on a project that's written in OCaml.

Need a real IDE with step by step debug, breakpoint, jump between files to inspect implementation. Tool to find all calls of a function etc...

I dislike dune syntax

D'une is not ready yet, make is pain

I write plugins for Coq, and it's becoming a pain to understand the different building tools, which are very fragile w.r.t. the specific OCaml version.

Ocaml documentation

Integrated/blessed build system and package manager, a la Rust

Improve runtime (std) lib

Make it easier to cross compile between OS/Architectures

Better build caching (possibly using cloud)

Quite a lot of complexity around build tools

Being able to set dune as bluid dependency

The newcomer experience

FFI is hell. There are a basquillion different tools, and none of them have the same defaults. (Dune and -linkpkg, I'm looking at you.) The ecosystem works great if you stick only to OCaml stuff, but...

An easier to install IDE

More stable opam upgrade, better dune integration with opam, dune maturity

add refactoring tools

Work to remove fragmentation and focus on a modern toolset in all documentation and official resources ie dune, ocamlformat, merlin, opam, vscode.

Modern documentation on the current recommended tooling setups, with comparisons to other language tooling and

more references for newcomers, especially around best practices

Same suggestion as learning pain point: provide a Racket-style all in one option for new users with common libs, compiler, opam, and a basic pre-configured IDE set up to work with ocaml (like DrRacket does for Racket). Easier to get new users on board that way, less chance a problem scares them away.

I would make the tooling much more friendly towards newcomers - it can be difficult to figure out, although the OCaml lsp-server is making leaps and bounds towards this goal.

Newcomer docs: one signle fullproof tooling-oriented onboarding place

Improve the visibility of well known and tested libraries/tools (tools is ok, libraries is still missing good pointers)

I would integrate ocamlfind, opam, and dune. Having to understand three different systems to do anything is a drag.

I would like something similar to Haskell stack that uses nix-style sandboxed library installations, and where the sets of packages have been vetted for internal coherency, the way they are on Stackage.

make dune easier to understand and less obscure maybe

Less boilerplate would be nice. Better documentation.

documentation and standardization

Sound management

Easier and more lightweight per-project package and dev env management

Building RPM packages is only compatible with Dune but not Opam Workflows because Opam mixes compilation and installation which is problematic for RPMs.

The opam cli is very confusing to me and I can never get it right. I mostly use esy instead but that is not complete either

make something as straightforward as the tools in rust

more basic dune example repos in an easily discoverabble spot. fleshed out a lot.

Better integration with the REPL. Easier code reloading without restarts.

Introduce more convention over configuration reducing build system boilerplate

Esy/reftmerr support for watch mode

- more documentation for dune, especially user- and beginner-oriented documentation

I'd quite like 'dune @fmt' to allow registering custom formatters, but that's such a minor thing.

Better support for several sub-projects sharing some source files in the same root directory

Ease the use of the ocaml top-level within the projects (dynamically load of library, improve dune compatibility

Opam and Dune are geared towards daily developers. For more casual programmers, they are hard to understand, in particular when they fail. We need *many* short tutorials (in the form of a curriculum?) to understand not only how but, above all *why*, to use their sophisticated features.

Better documentation for building projects with a syntax-based part (menhir / ppx)

A current book about dune would be very helpful.

Have more integrated and supported tooling Have more projects using dune 2 "sandboxed" workflow

Nothing

maybe dune and opam should converge into one tool, like Rust's cargo

Continued development on dune to be able to handle more use-cases. Better dune/emacs integration (e.g. live build status, jumping to errors, etc.).

I find Dune's documentation a bit hard to follow. There are the quick recipies and then the formal references. But, so far, I haven't been able to understand "its essence", for example, the reasoning behind some of its design decisions. In other words, I don't feel confident enough to write a dune file in a situation not covered by the quick recipes.

Better documentation, More beginner friendly

Dune (especially when producing JavaScript output for js_of_ocaml) could be faster.

I'd like a better debugging support.

Going from OCambuild to Dune is a terrible amount of work in many projefcts. There should be some form of feature parity and automatic conversion.

More tight integration between package manager and build system. Listing dependencies in 2 places, and under different names (needs some fiddling to figure out what to write in dune file given some opam package name) is confusing, especially for newcomers. Dune by itself is pretty awesome!

Sometimes can be hard to figure out how to do something in dune. The documentation is there, but more examples of not so common workflows would be nice. Occasionally you want to include external files or depend on external files to generate other artifacts.

Need a useful debugger–I'm reduced to print statements and endless builds in order to debug. In particular, there should be a built-in way to view data structures, to see defined global/local symbols. Handle polymorphism as well as possible (e.g. maybe let user specify type parameters at debug time). It would help to have a way to access type definitions programmatically so it's possible to write generic code to print data structures rather than having to craft a separate print routine for each data structure.

Simple, easy to write build system

better build system for large projects

Too many ways of doing the same - provide a set of good defaults

I only use dune, and for the most part it's great. It's sometimes hard to find examples for unusual use cases, e.g. running a bash script as part of a test.

Improve cross-compilation

Installing opam and dune is always a pain. I also found learning dune difficult. I would consider making the new user experience with these tools better

Benchmarking, refactoring, test coverage are blind spots at the moment that lack useful tooling.

It can be difficult to write your own dune file. Some tooling around that would probably be useful and make it more welcoming to newcomers.

I find it heavy to quickly build small experiments

It is still complex to build ocaml and opam/packages against an other version of the libC (tuning CFLAGS, LDFLAGS, gcc/glibc/...). Lots have been done recently but it is still a fight :) FYI We have to build/link against a very specific version of the libc, not against the system version.

Standard library is absolutely awful. Tooling lacks (language server doesn't work well).

More documentation regarding build tools.

Have a robust, and simpler alternative to Esy (Spago on Purescript is my favorite all-in-one tool so far)

Split Bucklescript and Ocaml into separate languages on the frontend

build tooling should be more unified and the documentation should be easier to follow. Sometimes is hard to figure out how to link the pieces, as they are addressed too much in separation or with old documentation sometimes.

Better support for Ocaml in better editors

dune files are weird

I wouldn't install a "secondary compiler" to install a build tool

compiler error messages can be arcane, difficult challenge to solve I know!

I wouid unify opam and dune

There are too many moving parts and manual configuration required to build. Compare this with more modern languages like Go and Rust building is simply a matter of running a single command with usually zero configuration. Every time I start a new project I need to look everything up again.

a better consensus on what to choose, and this choice must be in the manual!

Writing ppxes is too difficult. Tighter integration between opam and dune would be good. Opam is a bit lacking/confusing when used to manage dependencies for a privately used executable (where e.g. the maintainer and license fields don't make sense).

Debugger: allow to show abstract types in debug mode

1. faster merlin; 2. something like typeclasses (modular implicits?); 3. support for returning multiple source locations in merlin and making this work with xref in emacs

Improve merlin (merlin-destruct is not easy to use, renaming could be improved etc)

better integration into popular IDEs

enable cross-compilation, build the compiler itself with dune, improve testing framework for the compiler itself, other compler infrastructure improvements such as modular use of compiler components via compiler-libs, clearer configuration.

use Ocamlformat in more projects (I'm not being able to use ocp-indent well in VSCode); make a basic tutorial for dune explaining under which dune-file conditions which files of the project get built etc. The dune documentation is great to look up concrete things, but not that good to start understanding dune

Dune is not customizable enough

I'm not an expert on packages in the open source world but with npm packages are installed locally in node_modules which means projects are quite idependent from each other. But with ocaml I often find myself having to recompile the whole of core or async when some dependencies change. I think (?) they are system-wide which makes it a bit annoying.

Create a cargo-like tool that embeded opam, dune, odoc and some test/benchs tools

dune needs more compatibility with tierless programming.

Intro to build tools have to present

Easier installation for e.g. ocaml-lsp

I use opam infrequently and I typically get a bit confused by the command line interface and some of the terminology when restarting using it each time. It would also be nice to support light weight switches so I could easily have them be per-project, though I think this was already being worked on when I last checked.

Focus on web more. Focus on text and video tutorials.

Make ReScript and ReasonML even better!

It would be nice to have something something like cargo where there's a simple, opinionated CLI for setting up and building projects

opam is very difficult to use. I'd rather have something like cargo.

I'd like to use bazel, since I use it with all projects in other languages, but dune is quite good, so no biggie.

The build time could be faster.

Doing float-heavy work in OCaml is a hassle

The setup of these tools is very manual, especially for new or close-to-new programmers. I recall in a college course I took ~2 years ago, I was the only one to have successfully setup Merlin and was the only one who understood the compilation model enough to add new files. This has improved with Dune, but, to my knowledge, is still difficult.

make a ocaml debugger

I'd work on adding a good debugger.

Improve compile time scaling to large projects

I would prefer if reliance on new tools (such as dune) was less mandated and less opinionated. We should have a clear separation between the core language and a diverse set of generic tooling around it. 'It only works with dune' is an incorrect and discouraging ideological shift in my opinion

Add cross compilation support

Better ergonomics around records (less namespacing noise), better solution for ad-hoc polymorphism, nicer type errors

unify the tools under one thing (like go), if possible. utop should be ocamlc repl, e.g.

I'd make it easier to make temporary edits to dependencies (aka a duniverse)

dune + opam would be more integrated, faster, and easier, with better documentation (as inspiration: Haskell's stack, Rust's cargo) building tools, did not find a convincingly better tool than makefiles

Make dune less opinionated

Better official status for a single build system (OCaml manual points to ocamlbuild, which advises to use dune).

better support for window

faster builds? Maybe we should use dune. Better error messages.

having to use ocamlfind for one-off compiles that don't warrant something heavyweight like dune is very verbose... perhaps integrating ocamlfind into the compiler, so it's not a separate command would help

Warning and error flags are confusing. Dune has made a great job in integrating many of the tools but there are still quite a few concepts to juggle, such as ocamlfind, merlin, ocamlformat and opam.

Not sure if I would change anything.

Dune is nice but feels like it might one day collapse under its own complexity. I'm scared that we're falling into some local optimum w.r.t. build tools.

One command to run that does everything, thoroughly and clearly documented

Simplify bootstrap with the ability to install a minimal system (ocaml, ocaml + build tool, ocaml + build tool + opam, whichever can solve the problem).

Dune language is highly inconsistent and documentation often inaccurate/outdated

There needs to be clear messaging and documentation about which build tools to use and how to use them when coming from things like PHP's Composer and NPM/Yarn.

Improved editor tooling (LSP/merlin)–I know LSP is currently under development. Also, OCaml could benefit from a local-install, version-locking package management system like npm, poetry, etc.–for deterministic + declarative dependencies, instead of the global-by-default, imperative opam way. Finally, odoc is a great tool, but it would be nice to have an official documentation website for packages on OPAM; currently, every package needs to supply its own documentation website, or you're forced to build it yourself with odig/odoc–docs.mirage.io used to have pretty good docs, but that disappeared for some reason.

Dune is really quite good. I honestly wouldn't change a thing about it.

private opam repo is very hard to figure out

dune evolves really quickly, but the docs can sometimes fall out of date. (In general the docs are fairly high quality already though.)

ocaml needs full debugger

Dune solves a lot of problems in the ecosystem of building OCaml projects, but restricts the way that programs can be built and structured in ways that are incompatible with my old development workflow. Overall, it's pretty good, but it's configuration format leaves a lot to be desired, and external shell hacking is required far too often for a good build system.

Dune doesn't support packs correctly, which means we still need to use Makefiles.

start deprecating and removing things

opam on windows

People not adopting standards

Better support for static builds and C bindings generation

Standardized solution for documentation with executable tests as examples in docstrings

Simpler git clone, compile, install story from scratch

Ease building programs written in several languages (Coq + OCaml, Rust + OCaml)

Better IDE support

More tools working on Windows OS

Lockfile-style versioning with minimal opam involvement (as it's too slow). Currently I use either dune vendoring (fast but not mainstream) or opam pins (unacceptably slow)

Better support for gdb. Less churn over the years around the tools! Centralized documentation repo of every versions of every opam packages?

Better documentation and tutorials

Make a consistent language release with a standard library that people actually use and is documented. Make sure popular editors have good and equal support (emacs, vim, vscode) for all parts of the ecosystem. Define what is "idiomatic" use of the langugage and tooling. Most important: don't name executables with the DOS/Windows-extension ".exe" on non DOS/Windows platforms. Why is this so hard to fix? It signals a lot of nonchalance to newcomers. Also, it seems that all tools are single-threaded (opam).

The path to language-specific build tools is a huge mistake, that the Erlang community made with rebar3 and is turning away from. I would much prefer seeing the community adopt rules on top of ninja, that make integration into larger, multi-language projects easier.

Documentation is lacking. I am not sure if Opam (for instance) can deal with git submodules when installing my library so I cannot use it and I am also unsure how to get my project into Opam.

Would be great if there would be a build tool to compile and link in C++ code

Too many (not always working) solutions for building and deploying

Dune has been a big step forward, but it's intimidating for newcomers. The interaction between Dune and OPAM can be confusing for newcomers.

The documentation, and the standardization of the cross-plataform support.

syntax

OCaml lacks decent debugging support. OPAM lacks proper binary packages (with reproducible builds) (I know opam-bin might soon be filling that pain point)

Better documentation for corner cases where we need to invoke other tools during the process or break code into separate libraries. Examples will help illustrate. The current documentation seems like it has gaps that I could only fill in by finding a project that used dune the way I wanted to and borrowed their dune files.

dune should support eliom/ocsigen

Make opam and dune easier to use, cargo is a good example

The state of cross compilation and mobile targets is unclear.

dune should be more extensible / modular, not just a behemoth of random things that Jane Street decided needed to be supported

Dune is beautiful, s-expr pretty printing is beautiful, and then you have the language itself. ~Sigh~ The syntax needs help.

Make dune less assumption oriented

Improve OCamlBuild, which is the most simple tool.

More documentation for the standard library.

Waiting for multicore support. I never use the object system, which, from my point of view, could be removed.

Better support for native compilation on Windows

it would be good if it was unified more

More tool/doc to develop/debug ppx extensions

I don't know — need to try the latest build tools.

Single tool, had problems with dependencies, improved documentation

Not enough special-purpose libraries, yet way too many standard libraries with different APIs!

Intellij IDE support. Mathematica (Worfram Research product) build in OCaml (lets do it!). Keep in touch with great programmers like Leroy and ROBERTO DI COSMO, GIANAS, etc. Be able to have a job using OCaml.

Better debugger support

Better error messages

I wish Windows support was better, because unfortunately most of my PCs run Windows.

make it like Rust's cargo. There's a single tool to create projects, install dependencies, run the build, run tests, etc

Dune is pretty good as it is for my needs. Native FORTRAN support would be nice.

Dune is too prescriptive. Setting up workflows anticipated by the developers of Dune is straightforward, but custom workflows seem to need built-in support and sometimes the building blocks are missing.

See esy. It does a bunch of nice things that opam lacks. But esy is not the default used in the community. Need to converge on the best of esy and opam.

Dune works but is a weird tool. S-exp is uncommon. Needs better tutorial/docs. By not being explicit on how it considers or excludes files. The newcomer will wonder a lot. It took me quite an effort to get it right (I still don't know if I *really* understand it). Implicit is harder to have a good understanding... maybe I lacked a good explanation. I don't know. But I have struggled with it. Coming from JS world I am biased and in favor of JSON, YAML and now Dhall. But s-expression is weird. It is likely possible to make it friendlier to a wider audience.

I would like ocaml to be as easy to use as Cargo

merlin/language server implementations need to get a lot better.

Too complex

- More sophisticated build tools, e.g., snapshot like Stackage. - Native Windows support will make it easy to support non-dev users of our applications. - Searching with types like Hoogle

More investment in utop integration in vs code

I'm incredibly happy that the OCaml community has finally settled on a definitive build system with Dune, but I find the documentation of Dune to be very lacking – I often have to find example dune files to figure out how to do something since the official docs are either out of date or entirely missing the feature I'm searching for.

For tooling: better compiler errors.

Standardize it and make it work as expected

more low level control, unboxed stack allocs, GC scheduling, nicer profiling tools, cross-compilation, unicode strings & characters being first-class (like Go), merge ocamlfind functionality with ocaml toolchain itself for quality of life.

dune's syntax: welcome back to 70's... Cargo (Rust) shows the right direction to the future

The Dune documentation is inadequate, it only gives examples and doesn't explain how it actually works, making it difficult to customize anything.

Nothing really

Officially adopt one of the standard libraries (probably Containers).

The documentation! Documentation in OCaml is almost universally bad, especially for non-experts. It is dense. It assumes knowledge that the reader won't have. It explains complicated things before it has explained simple things. It is absent or not up to date. Very poor documentation is almost a tradition in OCaml.

Some examples of workflow on the official web site, using the different tool sets.

Dune could be a bit more flexible/extensible - for example, I often run into issues when building bytecode binaries specifically because the structure of dune forces me to use specific build rules.

Better orienting tooling for beginners

More doc and more "how to" examples

Beginner friendly documentation. Don't assume I am already an expert.

Just assume Nix is available by default. ;)

It would be good to have a more "automated" installation of a "reasonable" environment for beginners

official build tools but as optional

More statistical libraries

Standardize on one build tool

More documentation and example in various tools

A.3 What would you change concerning package repositories?

Active work on a more secure package distribution chain (signatures etc.) could be reassuring.

Mutable opam metadata is a bad thing.

Creating a package remains hard : documentation of opam description files is very hard to follow. A collection of small examples would help!

more high level libraries and batteries included. I often reach out for things like: proper epoll support, argparse, json, SQLite, configuration files and so on.

Add reliable mirrors for the sources of all packages for all versions

Create a opam search repo

Many packages need manual tweaking to compile on Windows

Use stronger hashes everywhere

Cross-compilation repos need love.

Improve documentation. More examples and tutorials

Better support for multiple repos

improving the security (supply chain); stating the immutability of released packages more precisely (maybe camelus checks)

Improve Windows support.

Improve the windows story

enhance compatibility between opam*

Make opam packages immutable.

needs more people paid to work on it

The ability to easily create mirrors for use at organizations. Interop with distribution package managers for non-ocaml software: generate pkg-config files, debian packaging rules, etc.

tooling is amazing. But it lacks 'getting started' materials, read code and apis is not beginner friendly

Basic data structures are missing. Available libraries are mostly not maintained.

I fear a little bit the python or java package model with many different librairies with very different quality and maintenance and bloating. When reading a python program with tens of imports it is difficult to understand the code (especially if there is a poor documentation). Probably some sets of standard tools would be good. Trying not to reproduce the Async/Lwt conumdrum would be good.

I feel like it's more complicated that installing something from npm. I have to think about what switch I'm using and a bunch of other things and I just want to do 'opam install x'

have a way to search opam and npm at the same time

Changing a package metadata after it's publish does not change it's version (packages are not inmutable)

Introduce also binary artifacts package repository

More opam tutorials PLEASE

Opam-bin adoption could help a lot.

- opam and allow local dependencies; esy is great, but I've had difficult to track issues. It would be great if the official package manager just supports by default flows like npm and cargo - package deployment - having to submit PRs to a remote repository to release packages is an obstacle preventing people from doing it. This has led to excellent packages like ReasonML's Rely being only on npm and accessible only through esy: https://reason-native.com/docs/rely/ - the editor setup is incredibly confusing - why is there ocaml-lsp and .merlin - what is the default and easiest way to get started

Add signing

like above. make packages truly immutable, enforce bounds by default (at least try to follow semver, but having a dependency without any bound on its version should be a code smell instead of the default). per-project build, like an opam switch, but without having to recompile+reinstall >1GB for the compiler and tools.

Having a contribution workflow less painful

make opam use project-local everything by default

Nothing, opam is great

Test more package combinations

Binary package

Nothing!

Use json as the confit language, don't gatekeep packages, etc.

I need opam full mirroring (with source archives)

Automatic local switches in combination with Dune.

Similar to my last answer, we need intro material that shows how to take advantage of the opam repository and how to use unreleased/development repositories so that new users can have a smooth experience. We currently assume users will understand why strict version constraints are important and understand that opam's complex solver is useful. We should be upfront and direct about why. A few sentence paragraph would help set expectations.

Make it possible to have multiple versions of a lib in a project

I don't know. I've never successfully used opam though. I always just revert to vendoring projects on github and then spending hours patching the projects to get compatibility.

opam pin breaks for directories that are not the root of the repository

Same compatibility between opam and esy

Generate docs for the opam repository and host it at docs.ocaml.org. Highlight best packages for common use-cases.

Just not enough libraries out there.

Documentation should provide more examples, provide non-monadic versions for people who prefer old fashion async

I'd like to see efforts from the foundation to fund specific projects (eg, grpc support, http3, etc) instead of hoping someone in the community finds time for this. That would help everyone that has some time, but not enough to do it all by themselves, join efforts in something larger.

Allow multiple package versions side by side. Installing two version of Coq with OPAM is a mess

no more package revisionism.

Less opam, more esy

I hope there is a more convenient way to specify the version of the library that the current project depends on.

Not much. Would be nice to speed up compilation times with something like distcc.

dune docs

Nothing

When I looked into creating an opam package, it seemed like a lot of work. This could be made very simple with standard automated scripts. Maybe this is better now, though.

Improve Windows support

I would not keep as many deprecated versions of the library. No one actually tested installation of these not up to date version so dependency constrained are often way under-specified. Further more, having all these choices makes the life of the solver hard and it ends up making weird/non deterministic one.

Probably my previous text applies here as well.

Curated option and uncurated package manager would both be useful

Too many "suites" / alternate stdlib packages

OPAM dependencies often trigger massive rebuilding (...dune caching was supposed to fix this, IIRC? I just stopped running 'opam update' on any regular basis lol).

Need package namespacing, stronger security guarantees for package archives

More structure. At the moment the namespace is flat and there are too many packages making it difficult to find packages you want.

OPAM as a tool needs some improvements (e.g., no way to do integration tests _after_ installation, docs need a lot of work).

More libraries

I would investigate reproducible builds to make sure all packages for a particular version of a compiler always build. I would also investigate possibilities for binary packages so that installation doesn't depend on a particular library on a user's machine.

improve security of package ecosystem, opam is great but seems to rely on the efforts of very few people keeping it running, allow namespacing/scoped packages

Not much, opam is a fantastic repository! It doesn't have too high of a learning curve for more advanced usage either.

A smaller and stabler repository that ensures stability, reproducibility and good solver performance

The repository via git involves a ton of files. This makes updates slow on NFS mounted filesystems

Build errors are difficult to solve

high quality repos, but too small in number

package management should be scoped to a project with no global state or need to setup an environment. npm gets this right

redo opam with a better design

Signing, immutability, easier submission process

This may be more package management, but I really like esy's model of a shared package cache (rather than switch per project, or a shared switch for all versions). However, it's definitely awkward in other ways than opam.

My experience is that, as time passes, my opam installations "decay" and, ultimately, I experience many errors, to the point of deleting a switch altogether and recreat it from scratch. Also, recompiling package dependecies at almost every upgrade or install is tiresome. It's also a big problem for poor students with low-grade hardware. A repo of binary packages would be most welcome.

Don't base them on Github (Microsoft).

Nothing

still a bit cumbersome to contribute packages, seems like a lot of toil on the ocaml.org side as well

I feel like basically every time I run [opam upgrade] a bunch of dependencies break. More stability would be useful.

Implementing the OPAM registry as a git repository is probably NOT going to scale in the very very long term.

Search all packages by function type signatures, like Hoogle

Better discoverability of packages

Allow to define Github sources in .opam files without forcing users to explicitly pin.

New OPAM packages often get stuck for a long time as pull requests due to the incredibly nit-picky and unstable CI, which is a huge deterrent for even submitting something in the first place.

Make opam more user-friendly. Lack of tutorials makes it hard to figure out what to do when you only at start of learning OCaml ecosystem. Compilation time for large dependency trees is demotivating at times. opam-bin is a great improvement in this regard. Debian, like many other Linux distros, solved the binary distribution problem, I believe opam can do that too!

OPAM is great. Still have to read a lot to make a package and release it

I have no complaints with OPAM and its package repository

I found that getting a package pushed to opam is very difficult. The tooling could use some work, and the workflow with needing to get review from a human on a PR makes it easy to lose momentum.

Core, Basic, JS confusing

There are too many steps to manually publish in opam. There are tools to streamline the process, but the process should be simpler.

no downgrading everything because the solver is dumb, no rebuilding everything because dune was updated

more community best practices

I can generally find what I need but opam needing to rebuild everything is a bit frustrating. On the other hand I really don't want to install npm and everything that comes with that in order to use esy.

better documentation

Improve security with package authors + transitivity dep check

Improve documentation of opam interface

I'd use Nix, to be fair. Not because it's the last cool thing but because I'm tired of all languages having their badly defined package manager. Opam is slow, unsafe, complicated and has a confusing CLI.

Most of the repos are outdated.

More coherent libraries, a single shared signature for monads, etc.

opam's user experience is very good, but opam packages can be changed after being published is crazy. Serious flag raised by security auditor, we'll need to stand up our own internal repo with unchanging releases somehow before we go to prod

There are essentially two worlds of OCaml packages: the Jane Street ones and the non-Jane street ones. Some kind of unification story would be amazing as it would avoid me linking in 2-3 standard library replacements when I want to have external dependencies and needing to consider whether to use LWT or Async projects.

It would be good if OCaml packages played well with the distribution's package management system, such that one can easily do 'apt-get install ocaml-lwt'. If these are kept up-to date with first class support, it would reduce the barrier to entry for newcomers

More beginner-friendly error messages

I'd make it easier to have small packages not fall out of date

have a more active community?

I had a hard time last time I updated opam (had to delete the default switch and recreate it after 20min of useless and frustrating tinkering).

I forget, it's been a while, but I did find it less developed than python/java/etc. Obviously there's no easy fix for that, and I have no small-niche-language that I could compare it to.

Make opam more intuitive and cater for the common work flows. I still need to google for how to upgrade to the latest compiler.

n/a

Add tags to opam package search, like Hackage's "packages by category" view

We need a good high level architecture (especially JS libraries). Currently it is either you install all or nothing. With lots of interdependencies of packages they become essentially monolithic. This tendency to make everything an Enterprise Monolith is a huge invisible threat to the ecosystem. Windows is essentially lost because of that - I can't find a standalone compiler build for this OS (either OPAM or nothing). This complicates bootstrap, packaging for Linux, and if one of the libraries has problems, say, on Haiku OS, the whole cluster can't be installed on this OS.

More documentation and beginner-friendly guides on configuring, publishing to OPAM, etc.

opam is very hard to integrate in Enterprise environment because by default it does not support lockfiles and install packages globally instead of per project.

debugger support

Opam is pretty great, not much to change. The review process for packages is not the best, but it's still far better than many other package mangers.

we need fresh and easy to read documentation

Opam is a disaster, I'm sorry, it breaks very often for me, not to say it is source-only tool

Add mirroring support, deprecate outdated packages more actively

Nexus / Artifactory or equivalent support

Hard to know which alternative library IS the Lost used

naming of the libs is confusing: the purpose of a lib should be almost obvious when reading its name.

Have more people contribute to nixpkgs

Better best practice guide. I’ve read a lot on how things work but don’t know when it’s best to use functors instead of generics.

better support for nix / guix

Better documentation

Very confusing with "switches" etc. You have to manually reinstall everything when upgrading the compiler.

It seemed great early on, but in the last few years, I have had to stop using opam because every interaction would result in breaking everything for me locally, particularly because something would get upgraded and the version of something else wouldn’t be available. I decided npm and gem were enough problems as it was, I didn’t need another with opam.

Nothing! OPAM is great, and a huge resource.

The update of the packages to support the last stable versions.

In OPAM repository, the packages descriptions are modified in place, which tends to cause multiple problems. I would rather follow the debian’s convention to create a new package version with the same sources instead. OPAM repository lacks curation, which is good. But it would be nice to have some kind of curated version of the repository with some kind of guaranteed maintenance, co-installability and minimal support

Better support for stable release of packages. Cargo-style packaging

opam should be available for Windows (as it is for Linux)

We just need more libraries and more active library maintenance.

Publishing to opam is very complex

The primary way to get packages is to use opam, which acts as yet-another-package-manager on your system. This is slightly gross. I don’t exactly have a good fix for it, though.

Publish package API documentation on OPAM web site.

It’s pretty good overall, no major changes needed for my work.

Nothing. It was great pain to submit OPAM PRs years ago, but now things get much smoother. Thanks for the OPAM repo team!

It’s a bit lacking

Download all and have a disconected local repository (I don’t know if that is possible) and compare it with other options (explained from the teoretical point of view.. not only tooling) know things like that is one of the adventages of having formal trainned, teoretical programmers in the community.

Having to make a PR to opam prevents me from wanting to release packages

More use of tags so it is easier to categorize packages.

Don’t understand what this question is asking? Quality of the packages?

Opam not isolating the environment makes it pretty bad. Esy does better. But it has it’s quirks too. Lately I’m experimenting with Nix.

Everything

A much more discoverable platform

Easy to use opam difficult to make opam package

It'd better to have snapshots like Stackage. We are trying to workaround that in https://github.com/na4zagin3/satyrographos-repo.

Better support for Windows and binary libraries.

I'm not the biggest fan of the opam vs. npm/esy split. I prefer opam in all my own work, but when dealing with projects that originated by people new to the OCaml community or are meant to target newcomers it seems like esy/npm are the preferred choice. It's rather annoying to have to consciously think about which repository a package resides in. In addition, I do wish that submitting packages to opam was as streamlined as npm.

Find some way to interact nicely with other language-specific packaging tools

Learn from Rust

Cargo, cargo and cargo :-)

I'm quite happy with opam, no real complaints

I don't know

There are too many packages for not so many functionalities: packages are often too small and functionalities are duplicated. There should be vetting of new packages.

Easier to publish and build from pinned, use module system for package resolution

Binary packages to speed install, light repo with latest packages versions

To have more documentation in the packages published in opam

lack of doc

A.4 What libraries are you missing the most?

Better numerical - though Owl is great progress

Graphical User Interface, specific hardware drivers

elastic, sentry, prometheus

libraries to embed the interpreter, argparse, unittests

debug print - printing data structures without specifically using ppx

GPU computing

Machine learning

Database drivers that are written in pure OCaml for easy build/deploys, while being maintained by a larger group of people.

numerical applications (Haskell's hmatrix)

gRPC

Proper unicode support, ideally integrated to the language. Also missing proper concurrency support.

cross-platform graphical application frameworks (cocoa, gtk, qt, ...)

acceptable yaml serialization/deserialization

Numerical analysis (Owl doesn't completely cover my needs)

A serde like library.

Cryptographic functions (Schnorr, etc.), polynomials

Belt

It would be amazing to have an Elm-style model-view-update library that worked out of the box in a web page.

Clickhouse driver

scientific, numerical analysis, physics

base, core_kernel

lock-free hash-consing

Web framework, SQL query builder

Modern backend web development ecosystem. There are great low level packages- async, lwt, httpaf, ... But modern backend web development's bar is pretty high when comparing across languages. It's not good enough to have an incomplete smattering of libraries to try to put together when building a modern web backend / microservice. See talks by Paul Biggar and Sean Grove for more info

I'd love to see more game-oriented libraries: graphics libraries, physics libraries, etc.

a webserver with all needed for production. I know there is eliom but I'm not happy with it. Shil may be a good one. Opium is awesome but lacks common production rocks

Web based ones.

SDL bindings for Windows

I can find libraries, but often they are missing features i'm used to

Libraries/frameworks for web application development, and database ORM libraries

Numerical libraries. But Owl goes mostly in the right direction. But it goes many in the Linear Algebra direction whereas I use OCaml a lot in Analysis (integration, EDO, EDP, etc.). For that the good old ocamlgsl mostly fit my need.

JSOO bindings to everything. I would switch to JSOO if there were more bindings

web scraping

HTTP 2 and 3

Creating XLS files

It is not always a lack of libraries, more of a lack of documentation. This adds up to the issue that most libraries are hard to find

More beginner friendly, high level HTTP server / web client

grpc

Functional ideas; e.g. prisms.

Recently gRPC, I am not sure about the quality of many libraries such as web server libraries or database clients.

I think good libraries for interfacing with web APIs have been very lacking.

Again, I think you are asking about public libraries, with which I have no experience. Internally, I have or build what I need.

a stable ppx foundation, or rather, a "derive" integrated directly in the language that won't break.

An SQL library (other than Pgocaml)

a performant, batteries-included web server framework would fill a big gap

a good library for data visualization

Data science in general

Pure fonctionnal programming à la Haskell

OAuth2, machine learning

Grpc, dynamodb, elastic search, prometheus, open tracing

Modern web server framework

The Async / LWT ecosystem split basically means that software compatibility is a coinflip

I can usually find libraries for a given protocol, database, etc. but they are often not actively maintained. More fundamental libraries (e.g., Core, Async/Lwt) are usually easy to find

I prefer to minimize external dependencies. I would like a larger standard library, though no syntax changes (main reason I avoid Jane Street)

Web API's frameworks/tools.

Data-processing. CoAP. Distributed computing. Web frameworks like Django/Ruby on Rails.

swagger doc generator

Google firebase

A good complete web library in the style of Django (integrating DB automatic management for example..)

native firebase

GUI

monad? maybe

high level metaprogramming tools

Deep learning

lilv

Windows versions

A library to use ipv6 multicast easily. a more complete "ocaml-vdom".

Better wen frameworks and associated libraries

Serialization, networking, cross-platform abstractions

A version of Zarith that doesn't depend on libgmp (for GPL x static linking reasons)

I only use a few libraries; moreso bindings for hardware abstraction; not missing anything major

LDAP, NNTP

vendor-specific libraries, eg google cloud

FFI generator

Unicode, standard database api like PHP PDO

Networking.

native toplevel, JIT top level, easy code generation in running apps

cloud orchestration libraries, they are relatively immature and/or outdated (k8s libs)

I've had to build my own PostgreSQL library because other ones just aren't sufficient, and

OAuth2, AWS SDK

Cross platform modern graphics library for GUI development

Library to JIT code that interfaces with the OCaml runtime. A compiler-as-a-library (with reentrant API). A good parser generator for LR(>1) grammars.

I would like more web development libraries and frameworks. The TLS bug with the ocaml-tls library was a drag; glad that seems to be fixed.

Web based libraries.

Grpc, rest frameworks, database abstractions

Sound management

Web development libraries

P2P, crypto

gRPC and OpenAPI

base

machine learning on gpu. scipy equivalents. scientific visualization and pub quality plotting.

Kafka

Regex, ergonomic Requests, stable plotting library,

Mostly web related libraries

I can't think of any specific examples right now, but often libraries are /incomplete/ rather than missing entirely.

A modern, complete, cross-platform GUI lib. The JSOO compiler is good but I'm still looking for *well-documented* web progamming libs.

GUI / 2D graphics

Android / mobile programming.

Web tools and libs, DB drivers

robust google api libs

Better support for visualization.

An easy to use HTTP client (i.e. capable of being coded in both synchronous and asynchronous ways). And, as a consequence, a comprehensive AWS client SDK.

HTTP server

I am missing more JavaScript libraries bindings for js_of_ocaml.

Qt

I would like to see more code verified in Coq be incorporated properly into real OCaml tools

Everything related to HTTP servers needs a lot of work to really get going. cohttp is slow, httpaf is young and buggy, Anonio's fork is better, but it's not published on opam. All the surrounding ecosystem around backend (micro)services in OCaml is quite immature, no tracing library, no GRPC, no bindings to popular key value databases like Cassandra. js_of_ocaml is great, but the ecosystem around it is severely lacking even compared to ReScript.

More UI libraries would be nice. Gtk is okay, but would be nicer if there were other bindings.

DB

Non-LWT HTTPS client.

GUI, scientific visualization

Some areas are covered, but the libraries are basic and often unmaintained (xml is a recent example).

The existing web server stuff is kind of a mess.

If I can't find it, I implement it, so nothing for now.

menhir, ocamlfind

Frankly speaking, I only use stdlib (and my own libraries and tools)

SQL libraries are missing or are too experimental, same goes for Graphql

mostly libraries to interact with other tools/softwares

good UI libraries

n/a

GUI

In general the choice of libraries is pretty small compared to my main language of choice Go. I find it quite common to find abandoned libraries too. Coming from Go, I prefer to have a large standard library that is consistent. Because of this I've gravitated towards the Jane Street libraries but I find them a bit overwhelming and poorly documented..

GUI tool kit

gui, plotting, numerical computing, basic email

A high-level web framework ala Rails. An IMAP client.

Cryptographic libraries. Ocaml-tls, for example, is scary to adopt because the underlying mirage-crypto library (and nocrypto before that) haven't provides support for key crypto primitives like ECC.

Databases, Auth. Multiple paths: async vs lwt, cohttp vs httaf. Oscigen could be awesome but it looks like a somthing nearly dead. Lack of docu tutorials, documentnation, examples.

Numerical analysis

A Qt-like framework

Certain libraries that tie into distributed computing frameworks like kafka,cassandra,spark are limited or not well supported

LSP bindings, asynchronous programming, high level utf8 string functions, HTML generation, Markdown parser, ropes/splay trees for strings

I use internal libraries and am probably missing out on widely-used public libraries.

Lots of libraries exist, but it's normally a big hassle to use them

Reliable, high-level (with arbitrary safe typing) and efficient message passing library (unlike MPI, which has buggy and incompatible implementations).

easy layer for almost anything

Cross-platform UI library

data science

Nothing in particular for the work I'm doing

n/a

Convenience bindings for popular web APIs (e.g. SOAP/XML), common data structures & algorithms like Python's itertools and collections

A feature-rich and consistent standard library

Better stdlib replacements, better asynchronous programming interfaces.

we need a curated list of the well documented libraries

numerical librairies

Decompression libraries, RPC

Modern cross-platform GUI

scripting

Rewriting and unification

HTTP2

1. Rust interop 2. More efficient libraries in general

Databases and Network protocols.

There are plenty of libraries, but finding libraries that aren't uneccessarily LWT-ridden and PPX-heavy is harder than it should be. I end up writing my own, simpler, implementations of things instead.

A good Ocaml only API building (like Opium but more than that) like Python's flask

Portable, low-level systems code. I mostly rely on Jane Street's libraries, but occasionally there are things missing in Base and Core_kernel.

For example, libraries to build full desktop graphical user applications.

A good SQL library. The various ones are too low level for classical use.

the standard library is missing resizable arrays and priority queues

Great web frameworks. Great DB libraries.

Web stuff

more/better numeric libraries

Jane Street's stuff, which was poorly documented enough that, as a newbie, I gave up on the language twice before just using the standard library.

GUI: proper Qt bindings, better bindings for GTK 3.

Better and/or easier data analysis libraries + plotting

easy to use web development

Last time I evaluated Ocaml, I was missing kafka, GraphQL, ergonomic web server, protobuf, AWS

Libraries for the modern web

AMQP support for version 1.0 (https://www.amqp.org/) (As far as I know only for 0.9 version is availabe). Also for Arduino or Android and more support for https://mirage.io/

Better instrumentation (telemetry, like chrome traces. memory, although it looks like ocaml has merged in support for one of htem)

google cloud (esp Cloud Storage and Spanner), Eliptical Curve, a unicode string module (the components are there but you have to manage them yourself), Honeycomb, rollbar, Heap, a user auth system for the web

Scientific libraries (there are a few but more would be welcome).

mssql

DOM api for browsers

Data analysis

Yes, a lot of great libraries exist. But many many more are needed. We've had to write a lot of libraries, which ideally would have already existed.

Web, application related, databases, graphql, anything specific is likely to be missing.

Everything

High quality database and HTTP libraries (libraries exist but they're much worse than the Python equivalents). Web server /middleware helpers. Excel document loading and writing. High-level database libraries like SQLAlchemy. A good (comprehensive) AWS libary like boto. Comprehensive functions for text encoding and decoding (i.e. CP-1252 to UTF8).

Libraries to make a full cross-platform GUI

Web platform, database, no-sql

very poor in physics math numeric computation

File format libraries, e.g., YAML 2.0 and Dhall

GRPC support

Even in spite of recent work on http/af, I don't think OCaml's HTTP libraries/web servers are quite adequate for industry use. Likewise, using SQL databases is often a pain in OCaml.

Probably vulkan, but that's quite niche.

HTTP2

unicode handling & normalization

It's less libraries that are missing then libraries that are missing features.

WEB LIBRARIES!!

grpc

mostly just an established Rails/Phoenix-like web framework

I miss the diversity of python libraries, but even if there are libraries i find that documentation is very sparse and it's hard to figure out how to use most unofficial libraries

GUI (2D, 3D), or GPU primitives (just the lowest API, vulkan stuff) then building libraries should come from community (sharing happily personnal work)

A Computer Graphics focused linear algebra library like glm

Up-to-date GUI libraries (Qt, GTK)

Web related libraries

GUI library

statistics

Full-feature web framework. Ocsigen seems promising but found it hard to set up, understand, and use (~2y ago).

A.5 Tell us anything!

Thanks for your efforts

Documentation in form of tutorials for common tasks and workflows would be a great improvement.

thanks for taking the time to do this

Communicate as much as possible on OCaml!

Really appreciate everything the OCamlSF does – thank you :))

Please make the tooling for build and packaging more beginner friendly

Thanks for promoting this amazing language!

OCaml is a great language for industrial applications! The fact that my employer does not adopt it on a broader range purely is the low qualification of its staff.

I really hate the PPX stuff. It is only appropriate for mostly trivial stuff, and it is hard to use because the language does not have builtin support for quotations/antiquotations. It is possible to write such tools yourself, but it is extremely hard to maintain due to the constant changes in the OCaml AST.

- Take a look at Kotlin's syntactic sugar for DSL building (extension methods, infix functions, lambdas with receiver, etc.) :) - One thing I'm missing about OCaml is tutorials for language features. I found a few blogposts about writing ppx and using GADTs, but I still don't know exactly how to use them (especially ppx). With Java/Kotlin you can google anything related to language features and find loads of helpful tutorials explaining the features using simple code snippets. (e.g. searching for "java self" yielded this nice post: https://www.sitepoint.com/self-types-with-javas-generics/) How about having a few devs write some tutorials for complicated language features and post them to the ocaml.org website (which is very barren compared to https://kotlinlang.org/docs/reference/ for example).

Please continue making multicore OCaml a reality! You folks are doing a really great job. (Hope this is not wrongly addressed.)

I most need better documentation and tutorials, with lots of examples. Much of the contemporary documentation is exceedingly high-level and abstract. Many libraries just provide type signatures. I generally appreciate HowTos and concrete examples that I can then modify. I am concerned that many people in the community seem to have Haskell-envy and want to push OCaml toward a more purely functional paradigm. I don't want that. I use OCaml because it's a practical, multiparadigmatic language and because I still like to use print statements for debugging. I would also love to see a robust network of blog posts spring up where people were documenting their work, presenting examples and code, etc.

All of what I use ocaml for is personal things, so I don't have any perspective on what would be good for a more professional group.

thanks for setting up this survey, looking forward to see the results

Thank you for doing this :)

I love the language, it's shortcomings are minor compared to other toolchains / languages.

I feel that OCaml isn't inherently "too hard to learn", but is made so partly because of the various warts and culture resulting from a long history (which I don't really think is the fault of anybody). I would imagine something like the approach taken by Rust would probably make the language much easier to approach.

We really need better native Windows support.

Thanks for all! OCaml is awesome. But your "pain point" question doesn't even include the real pain point — it's way too hard for a beginner to install OCaml!

Greatest language of all, please continue, thanks

I would love to OCaml, but I can't use professionally. Also, optional dependent typing would be awesome. Another thing, I can't use js_of_ocaml because the runtime is huge!! (19kb for an hello world) But this is just me complaining, the people working on multicore ocaml, base, dune and the compiler are awesome. Keep up the great work!

OCaml surface syntax, once used too, feels great and plays a major role in my appreciation of the language.

One of the main problems has been that OCaml tends to change incompatibly between versions, which breaks things quite often.

It is great technology. Focus on the laymen without a CS degree. If they can make useful things with it, it will go far. There is no reason people should be using Go when Ocaml exists

Thank you for this awesome language !

OCaml is my favorite language!

Ocaml internship or job opportunities for many beginner irrespective of their academia are minimal.

I *really* love OCaml. I think the syntax for ReasonML is really good as well and opens things up to a lot of people that are put off by the StandardML-ish syntax. Thank you all for everything you do!!

While I don't think it should go into a full-blown "Batteries included" policy, it would be awesome if the Stdlib was complete enough that one wouldn't feel pressured to use a 3rd party replacement. Also, the Lwt/Async schism sucks. Here's hoping the upcoming multicore + algebraic effects will be the excuse for the Stdlib to include some sort of green-thread API that makes everyone happy and forces Lwt and Async to be deprecated.

Windows

The main issue I've found regarding adoption of OCaml is a general preconception that it's alien and very hard to understand. There is some truth in that it's different from the popular languages and there's a learning curve, but the concerns are blown out proportion. (I've found rust harder to learn and use)

Concurrency via effect handlers sounds like joy.

Prefer more development in things other than multicore. Multicore isn't very useful for me especially with the read barrier

OCaml is an incredible language (otherwise I would not have pushed myself so much to use it), but there are so many pain points that make it a tough sell compared to other languages: alien syntax (somewhat mitigated with Reason), lacking documentation and absence of beginner tutorials, no docs of good practices, weird switch concept for dependencies instead of local dependencies like npm, cargo, paket and etc., no "recipes" and solutions to common problems (builds, CI, patterns) - I feel I'm reinventing the wheel every time, no clear way of how to structure applications (patterns like dependency injection and their replacements in OCaml land), not backed by any of the big companies, was confusing to setup the editor (something you get out of the box with more commercial languages). Make it dead simple to start using OCaml and be productive. Attract more people with a diverse background. A healthy ecosystem needs people with different backgrounds, some will be brilliant academics developing the type system, others will write beginner blog posts and tutorials to attract more people, design nicer websites and etc. Before someone becomes proficient in a language they will start as a beginner. Being a beginner of OCaml is absolutely brutal. OCaml needs a branding. Make the value proposition clear - why should anyone spend their time learning a new language and ecosystem instead of using a more established language and building value for customers. What problem does it solve better than other languages and why does it deserve its time. How will it benefit someone giving it their time? Having clear value proposition and benefits also makes it easier to users of OCaml to sell it to others. I would have never tried OCaml if ReasonML didn't sell me the dream of type safety, native performance and variants. ReasonML also had nice easy to get started projects. These things are not enough to stay in an ecosystem, but are enough to get you started. Check Rust's homepage https://www.rust-lang.org - it explains Why, has recipes for building command line, web, network, embedded applications and has success stories from users using it in production. Even if these are very American, they could be dialled down, but these are effective to get started (assuming there are actual docs and tutorials to keep newcomers productive). Take examples from other languages - Rust has great docs, beginner tutorials, package manager (and hype). OCaml OOP has exceptionally ugly syntax, which is fine since it is discouraged, but show OOP replacements, explain why the module system might be better and how to solve common problems that OOP solves AT SCALE. As someone working on a 50 million loc code base and reading passing functions to other functions to replace DI: https://fsharpforfunandprofit.com/posts/dependency-injection-1/ just sounds completely ridiculous and does not inspire confidence that this is an entirely solved problem in functional programming. OCaml needs success stories.To sell the idea of OCaml to someone I need to explain to them what problems they can solve better and share success stories that would not have been possible without it. For example comparing the performance of Reason's OniVim with TypeScript's VSCode. Sharing the story of a company that is running web servers with OCaml with limited hardware, avoiding common bugs and etc. PS: Writing OCaml feels so good, but please fix the basics, one needs a very strong dedication to persist through the initial learning stages. This is not right.

Stay productive! It's amazing to see how much is done from very few people . :)

Thanks for everything.

I don't have enough OCaml experience to answer all questions properly. I hope it still helps.

another pain point is that RFCs/issues can linger for years, so it feels like it's very hard to contribute to the language (unlike, at least as seen from outside, contributing to rust) or even significantly to the stdlib.

Thank you !

OCaml is scary for who's don't try it!

I answered modular implícita because that's what I want, but if our goal is to make OCaml easier to use, I think redesigning the surface syntax would be better. For the second q, I answered documentation for libraries, but docs for the core language, build tools, and package management all need a lot of work. I actually find all of that stuff sufficient for me (I just looks at library source code instead of reading docs eg) but the lack of docs is kind of embarrassing.

this survey is a super cool idea, thanks a lot!

Would be nice to have developer tooling that guides newcomers thorough the learning process and helps agvoiding basic mistakes. Thinking of Elm as an example.

OCaml documentation, line Linux manpages, are often great examples of clear technical writing but are not as approachable for newcomers. A lot of work has been done to make the language more approachable (e.g., through books such as Real World OCaml), but not as much work has been done to make the tooling as approachable. The dune documentation, for example, only has a few quick start examples – it lacks a tutorial that guides you through the process of building a somewhat sophisticated OCaml project from the ground up, w/ best practices etc., showing how to integrate it with opam, etc. Something like that I think would be very high value.

Coming from a day job writing C++, it's hard to write OCaml programs with good locality that don't allocate too much.

The path of going from having no ocaml setup to being able to clone and build a project and have working autocomplete and what-type-is-this in the editor is too long.

Needs more interaction with institutions to teach OCaml instead of other programming languages.

We have a wonderful suite of language tools. We need to simplify and centralize use of these tools to make them more accessible and easier to get started with. Thank you for running this survey! We've made incredible progress as a community over the past ten years. Thank you for helping to guide that progress going forward.

The OCaml software Foundation should help small companies. It seems they are only interested in Facebook or Tezos.

- The errors that come out of the OCaml compiler are not helpful. 90% of the time, the message is less useful to me than the error message location, and I'll just navigate to the error and stare at my code until it compiles. Compare this to Rust, where the compiler errors are good enough that you can learn how to use the language just by reading the error messages and clippy lints. - Merlin provides great completion 70% of the time, and the other 30% of the time, it just doesn't yield any completion results. I understand that it's hard to do completions when the file doesn't parse, but files-that-don't-parse aren't exactly an edge case when you're typing into the file... - There's no tooling for refactoring. - There's no tooling for debugging. - Backtraces basically just don't work, especially if you're using an Asynchrony Monad. - The Async / LWT ecosystem split is very destructive. Any efforts to repair this would make many parts of the ecosystem interoperable - Having a WASM backend would be great

Is anyone actually using the object oriented features? Can we get rid off them?

I would really focus on getting new developers started with OCaml, especially ones who are proficient in another language but want to see how OCaml could help them. At the moment I see skilled people discouraged by minor getting started difficulties. I think we should count and minimise the number of keypresses needed to get started with an OCaml project, using libraries from opam and with VSCode integration all set up (bearing in mind that people start from Windows installs / homebrew packages / debian packages etc etc)

A native debugger is dearly missing (from the survey and the ecosystem)

Kilroy was here.

I really think the biggest barrier to adoption is error messaging around type inference, especially when it comes to the value restriction. Experts know how to navigate these errors, but newcomers find them very confusing. I know this is a hard problem, but there is a fair amount of research on this recently, and I think there is a lot of improvement possible. A marginally better solution with nothing fancy involved would be, in addition to communicating the type error, listing a few common reasons why one might encounter this type error, or linking to a resource on debugging OCaml.

Thanks for this survey. Looking forward to seeing the results.

Thank you for this survey! OCaml needs better promotion and ad like Scala or Haskell.

Typeclasses would be amazing, I'd never look toward haskell again. A built in deriving mechanism would be great as well (or finally implement hygienic macros)

This survey beats the python survey I did 2 nights ago.

As a beginner, I am very grateful for the opportunity to communicate with the community. My biggest wish at the moment is OCaml Integration, sorting and guidance of community ecological information.

Overall, I enjoy working with OCaml. It's the lingua franca in my research lab. But I miss being able to distill the essence of my programs into crisp, domain-specific formulations, the way I could with type classes or more general metaprogramming facilities.

Make a dedicated Youtube channel that teach people ocaml from beginner to make real projects, and also make an updated available jobs list for the new ocaml programmers

Reading documentation of projects using functors is horrible.

OCaml language is easy to pick up, not the toolchain. Try in browser should be wasm'd

Honestly the biggest hindrance to using Ocaml in our usecase is not having a mirror of opam on our offline development environment.

As I saw in a tweet, these last questions are like being in a candy store and allowed to pick only one thing...

Could GC be optional?

I think the two things that make it more difficult for me to convince others to use OCaml are (a) that the best introductory book, RWO, is based on a set of libraries that are idiosyncratic, with confusing documentation, and (b) that OCaml's syntax is so non-mainstream. It's not that I've tried to convince anyone–I think it would be difficult, so I haven't tried. I know there's disagreement about whether syntax is actually a stumbling block for newbies, and it shouldn't be, but I believe it is. I liked the Reason project for this reason, but over the last year it seems as if Reason or ReScript whatever it now has become is even more unstable as a project. One can't even tell what the correct website is. I was reluctant to use ReasonML to sell OCaml before, but at present it's just irrelevant for that purpose; it seems to be beyond bleeding-edge.

Continue your work. As an old ocaml user (since v1 in 1997), I appreciate recent advances but I find sometimes that the language is becoming more complex nowadays.

In the "Too hard to learn" I would say it's not entirely OCaml's fault, just typical FP not being yet mainstreamed.

Amazing progress in tooling in recent years. Excited for multi core and algebraic effects. The wider industry definitely isn't aware of how practical Ocaml is and to often viewed as very esoteric.

Lack of Windows support

I've not been following closely, but I'm bit worried by what I've seen/heard about algebraic effect handlers. Handlers that wish to call the continuation zero or one times are "already" covered by callbacks and exceptions. Handlers that can return multiple times seem... quite niche. I'm pretty sure that manual manipulation of continuations is the functional equivalent of a "goto", i.e. to be discouraged in 98% of cases. The effects typing system seems good, and effect handlers might be a good solution when buried inside very technical libraries... but ocaml users might be better served by just improving language & standard library support for slinging Result types around (using error cases as a form of type-safe exception).

Something not in the language features list: metaprogramming / multistage programming. There is MetaOCaml, but I always stick with the base language. And maybe modern expectations for tools/IDEs are not so compatible with metaprogramming.

Hi there ! I'm a student in the French "classes préparatoires aux grandes écoles" (CPGE), in the mathematics, physics and computer science section (MPSI-MP). In this section there is the possibility to take the "option informatique" which is basically a more advanced computer science class. The entire cursus can 100% be done using any other functional programming language, but OCaml is the language of choice in what i think appears to be every school. While OCaml is quite a good language, the entire installation process is such a hassle, especially for beginners. The student and teacher community knows this very well and the first part of the cursus is to help everyone get a properly running OCaml development environment, which isn't easy task in most cases. Although I'm talking about a specific OCaml community, I think a much simpler installation process / first time user experience could help the entire OCaml community, by enabling more new users to join, and begin quickly.

Ocaml needs some hype to get better adoption in industry. Maybe effects and multi-core might provide that hype.

If I had to pick one thing to upgrade one thing in ocaml: a real IDE installed in a single command line. (Which is not emacs based)

OCaml rules!

Code in unicode would be great for numerical/mathematical purposes! In OCaml I miss Julia and in Julia I miss OCaml.

The language and ecosystem is generally great, but some critical parts seem to be driven by either a single or very few people, such that if they stopped contributing, would cause serious issues. I think it would be welcome, as an industrial user, to see a community strategy for how to identify and define contingency plans for those critical libraries/projects without inflicting undue burden on the people creating these important pieces of the OCaml ecosystem. (somewhat similar to how ahrefs moved yojson to ocaml-community)

OCaml really needs better documentation - it's such a challenge for newcomers to figure out how to use comprehensive libraries like Core, Async, Cohttp, or many others because of a lack of solid documentation, and they're left to look at projects on GitHub that have often been left defunct for years. Documentation standards in the community and higher expectations would go a long way in making the language more accessible.

Non-standard syntax, difficulty to create a UI / web, and lack of stupid libs prevents adoption by "weekend programmers" who are useless, but drive numbers up which in turn brings more enterprises

Thanks for doing this!

The one language feature I would really wish for is dynamic types (which would provide deriving and so many other things (I know this is a very complex feature but you're asking for a wish)) Overall I feel like the OCaml ecosystem is headed in a very good direction at a pace that I enjoy.

For this question, I really want to say "mainstream editor support e.g. VSCode or JetBrains IDEs" If one piece of the ecosystem could magically be made state-of-the-art, I would ask for

I've only been using OCaml for a see months now (after having written hobby project a game lobby server in F# last year), and I am really enjoying it. So far I've ported a simulation model I used for a side project in my PhD research from python as a learning experience, and currently I'm working on a Matrix chat client. Most things are going very smoothly, and I find ocaml to be quite ergonomic. The documentation and learning/discovery experience for some things has required a lot of searching around. Like when I have questions about Ctypes that aren't covered in the RealWorldOcaml tutorial.

After 12+ years of OCaml development, I have mostly given it up for JVM-based languages due to the poor IDE support, poor debuggers, lack of reflection in the runtime, and most of all, the lack of recursive dependencies between files. The JVM is slow compared to OCaml, but the ecosystem is hard to beat.

OCaml is an incredible language and I could not be any happier that it is out there. I would love to see it become better known and would like to start teaching it in my courses.

For starters debugging ocaml code in prod is not easy. It's almost impossible or at least not that I know off to debug a static exe without log statements. Call stack dump of an exception would be super helpful. And no bytecode doesn't count

On one hand, it's great to have so many features in OCaml, on the other hand, we end up with a few important projects that use way too many of the super advanced features and it means ending up with something really hard to understand by most. For instance, modules can too easily end up being extremely complex (use a few submodules with a few includes and a few functors, et voilà, enough to get most people lost). Use GADT, you get there even faster.

phenomenal language. relative to others poor (but vastly improving) tooling and poor documentation

There's a lack of focus on the lowest-common-denominator user. One click, all platforms, code snippet on the homepage, get a webserver/GUI/console app running. No large step to take that and expand it. Not code-snippet or example based, but good defaults and more things that "feel familiar" for developers from other ecosystems.

Aspects of the language and ecosystem are driven by extreme demands from Jane Street which have too little upside for the rest. Features like files as functors or recursive modules are inviting software that is too clever for its own good. PPX is most important for type-directed derives but creates maintenance and dependency problems. It seems a language feature is missing here and we have huge outside complexity as a result. Opam and Dune are the two best things that happened in the last 10 years. However, submitting to Opam should be easier for the common case. Let me submit a project by submitting a github URL on a web page and not deal with crazy tooling or cloning a database first and make a PR to it.

Thanks for this survey! Despite all my criticisms, the work achieved in the last few years is great!

I also want built-in deriving

Support of floating point numbers is not satisfying for numerical experts (essential features of IEEE 754 remain unsupported, as rigorous pattern matching support, pretty printing, etc.).

There are a lot of features that could also be added that weren't mentioned in the Burning Desires list above. I'd like: - first-class patterns - exception safety guarantees (either in the type systems or through attributes) Thanks for making the survey!

I love ocaml but its obscurity has caused me to pick up rust recently, simply for the pairing of a good, efficient language with a thriving community.

Thank you to everyone involved in the community. You all do an incredible job.

It's not bad to have a small community of elite programmers.

The teaching experience is unfortunately rather bad for students. Dune is hard to understand for them. ppx_inline_test is great, in principle, but it sometimes fails without proper explanation (and we aren't even able to reproduce bugs). We would also need students to be *one* opam invocation away (say 'opam install ocaml-teaching' + VS Code) from being able to use OCaml, which is unfortunately far from being the case. From one year to the next, Ocaml and packages change too much for teaching. So the aforementionded single opam invocation should also ensure stability (possibly by fixing the version of odwnloaded packages). We also need binary packages because some students do not have efficient hardware available to them. The VS Code plugin is very good (with ocaml-lsp) but here also, some useful features are yet to be realized (e.g. simple refactorings, generate an .mli out of an .ml...). On a different topic, "advanced" techniques in OCaml (e.g. applicatives, monads, etc.) are mastered by good, daily developers, but there are no good tutorials and introductions for lay men. Said otherwise, the step from basic, good'ol functional programming to more advanced techniques and patterns is not documented.

Glad to have answered as honestly as possible. I love this community, keep up with the great ideas! Thanks for the survey.

I would remove the refernce to OO from the website. I think it would be easier to promote OCaml as a functional language. The object system is rarely used anyway. And the TLS certificate should be expanded for "www.ocaml.org".

Do we want OCaml to take over the world? If we do, I think we need to attract more core people who are die-hard Windows users. It looks like it's being treated as an afterthought right now (though changing). I'm talking about first class opam support, debugging, Async, and a shiny IDE. *I'm* totally fine using (vim|emacs) + merlin + ocp but I do get a bit jealous when I take a trip Java land and fire up a fancy IDE or run Visual Studio Code on a first tier language and see how full-featured they are. People from the Windows world will expect this kind of shiny seamless integration when they try to use OCaml, and it'll be missing, and they'll feel like they stepped backwards in time. Not a good sell for an otherwise advanced PL.

Thanks for doing this!

Albeit all its downsides, OCaml's history and persistence are heroic. Plus some of the software developed in it is very impressive. For example, Mirage is extermely interesting. But the current level of fragmentation in such a small community is tragic. For instance: the multiple standard library extensions/replacements or the BuckleScript/Reason departure from "trunk" OCaml.

A decent source code debugger is lacking (or I'm not aware of one)

It would be interesting to "formalize" and streamline more the transition of OCaml newcomers from the JavaScript world: - from Reason / BuckleScript (get started with OCaml while still using JavaScript tooling like package manager, learn about types, inference, etc) - to js_of_ocaml / OCaml (reach proficiency, access full OCaml ecosystem, learn Dune, opam) I have the impression from personal experience that BuckleScript can play a critical role on increasing the size of the OCaml community by an order of magnitude, but right now BuckleScript and js_of_ocaml are presented as competitors of sorts and there's no connection between both.

Please coordinate infrastructure and tooling more with the Coq community

There are too many ways of doing things - many stdlibs, 2 async libraries, 2 package managers, 2 JavaScript compilers. Nearly everything is largely incompatible. Developers have to suffer because many project owners are scared to depend on something, and ask to avoid using libraries or to vendor them. Vendoring is not as automated as it is in Go for example, stuff has to be copied manually, and name-mangled in case of libs. IMHO it would be awesome if something like Jane Street's Core was bundled with the compiler, included first class support for HTTP stack, gRPC, and had all the ecosystem depend on that instead of wasting time on vendoring libraries. And have all of that work on JS out of the box! Go as a language is nowhere near OCaml. But the ecosystem is largely converged around stdlib types and is thus fully compatible, and this is really convincing to select Golang for your backend service. Using OCaml there is a story of struggle and pain.

Thanks for all the work you do on OCaml! I've really enjoyed using it the last decade, and its only gotten better. Can't wait to see what the future holds :)

The statement in https://github.com/ocaml/ocaml/blob/trunk/CONTRIBUTING.md "We currently have more contributors willing to propose changes than contributors willing to review other people's changes, ..." may be true, but it is also quite discouraging. Recently I spent several weeks creating my first significant PR, which several people said would be very useful, but I have no idea whether or when (months or even years from now) it will be reviewed, let alone merged. It's been sitting there waiting for a while now. Eventually I may have to conclude that it's a waste of time to try to contribute to OCaml (and that the language is not fully open source). How about making the list of core contributors (reviewers) public and readily accessible, along with what parts of the code they cover? It would help to define reviewer teams (e.g. those who review library changes) and to automatically associate PRs with with the relevant teams based on the files modified. Maybe let submitters set labels on own their PRs to identify what part of the code the PR affects, which would make it possible to search for all PRs that change libraries. It would help to have a way to get a little feedback on ideas for PRs from potential reviewers up front–in particular whether the idea is interesting, worth pursuing and has some likelihood of being accepted in a reasonable timeframe. In my experience, getting such feedback usually requires direct contact with the relevant potential reviewers. Also, I hope you guys think about how ways to get more reviewers.

Thank you for your work!

Language features are the most interesting for me. Easy to get set up, and sample projects showing steps to build and release libraries would be great

As mentioned, my main pain point is the lack of well-maintained and documented libraries (with the notable exception of Daniel Bünzli's libraries whose documentation is fantastic). Also while Lwt is a great project, requiring it in a library makes that library unusable in non-Lwt legacy projects, so it's nice when libraries consist of a core library with Lwt and Unix IO child libraries.

I had a really hard time choosing a burning desire! All of them would be great additions to the language. I chose type classes in the end because that is what gets in my way the most when using OCaml, which is for basic scripting and for interacting with Coq.

OCaml is great!

OCaml is great. Increasing its adoption requires coordinated marketing to potential newcomers, rather than addressing specific technical deficiencies.

We are weak on the machine-learning front in terms of libraries. However, there are recent bindings to python libraries (like sklearn) so this should be one way out.

OCaml is my first functional language. I've been doing PHP+JS+HTML+CSS and C++ many years ago. Was working with Python as well. More recently I'm doing Puppet and lot of Bash scripting. I find learning OCaml quite painful, because the documentations are way too basic. There are no examples, snippets, best practices in most of the cases. The community is also small, so finding something relevant on Stackoverflow doesn't happen most of the time. Most of my coworkers at the company learned OCaml in their university years and their coding style is way way too advanced for newcomers to understand and use as a learning example. I finished the 6 weeks OCaml mooc, but I feel stuck at this point due to having no direction and no helpful documentation to progress learning. E.g. I'd like to get to know the libraries, but if I just read a complicated function signature, it doesn't tell me much. I'd appreciate examples about the usage of core and user library functions.

I think the syntax of Ocaml is great, and while many people like Reason, it makes doing things in Ocaml that I do often really painful or ugly, such as custom operators. I also think what makes Ocaml so great for me might make it less interesting to the wider audience, so I'm not sure I would like Ocaml if it did whatever it takes to become a popular language.

OCaml as a whole seems to have improved a lot over the last decade, I'm very glad for it and hope you can make it even better

It would be great if the byte code were more stable and compatible across point releases. It's potentially a great way of providing multi-architecture, closed-source releases.

I found it interesting that the state-of-the-art question doesn't have the standard library. To me that's the (1) worst thing about OCaml.

I use OCaml in research, in theoretical CS and linguistics (e.g., to express semantics of calculi, to prototype calculi, to evaluate calculi's expressions, etc)

Very hard to interoperate with other programming languages.

anything!

Have a nice day!

I miss a full featured degguger with GUI

I really want to like OCaml and it always draws me back because on paper it ticks so many boxes.. Pragmatic functional language, native and fast compilation, stability, backwards compatibility. But whenever I use it I die by a thousand paper cuts. * poor documentation and lack of libraries * fragmentation in async libraries, core libraries, parsers etc * dune is confusing and too manual to set up * needing to learn random ppx syntax to makes things more ergonomic * cross compilation is difficult * exceptions. I always have an underlying feeling of unease that even if everything type checks some code path will throw an exception I need to catch and this really spoils a lot of the advantages of the type system. I really appreciate everything the OCaml foundation is trying to do but without a growing community I don't see it ever getting the critical mass it needs to solve the issue I mention above. It needs a compelling reason to be chosen over Rust or Go on the backend in order to grow. It can compete with Rust in terms of ergonomics and Go in terms of expressiveness but without a community it will continue to fall behind. I'm hoping that multicore will generate some fresh excitement and bring in a new wave of contributions. Given all of the above, I still generally enjoy using OCaml for small personal projects but I could never in good conscience choose it for a production system that needed to be worked on by many developers over a long period of time. There are simply too many unsolved problems that other languages have already solved and my role as a developer it to bring value to my employer, not reinvent the wheel.

windows support prevent many users

thanks for OCaml! I'm having fun with it. The community is sometimes not very open to objections, but I guess that's everywhere the same

Debugger: allow to show abstract types in debug mode

Among my friends that I encourage using OCaml, the language is considered as "not for industry". I do not know how this image could be changed...

Complete lack of marketing and communication from the core OCaml team and the OCaml software foundation seems to do absolutely nothing about it. It is time to have professional communication about OCaml. In industry it is very difficult to be taken seriously if there is no strong and clear line from Inria and OCSF.

I am excited that I got into systems programming thanks to ReasonML/BuckleScript. It's fairly straightforward thanks to esy to write some system tools which are waaaay faster than their JS counterparts. Now, even though the rebrand of BuckleScript has put a wall between the OCaml and BuckleScript communities, we still learned how to use a meta language after all. Even if ReScript or Reason would die (not gonna happen), I'd still be able to write OCaml or pick up F# pretty easily.

I teach 400+ students OCaml every year and semicolon syntax / precedence is the number one stumbling block for learning the language. Polymorphic structural equality (=, <, etc.) should be phased out when typeclasses/modular implicits are added.

I really like OCaml. Don't get me wrong. It is only so hard to build something bigger then a small script in it. Struggle between choosing Async and LWT or cohttp or httpaf...I can choose LWT and Httpaf but then if I need webcockets I have to use cohttp etc.

OCaml is much less Google-able than other languages, particular on sites like Stack Overflow. I wonder if the somewhat deficient standard libraries are part of the problem, because many industrial users just use their own replacement for the standard library.

Whenever I used OCaml for personal projects outside of work, I found that they broke a lot because so much was changing out from under them, so I just don't use OCaml outside work anymore. (At work, the person who breaks my code has to fix it.)

Debugger! Debugger debugger debugger debugger!

most people have no idea they have problems ocaml can solve

The lack of (apparent) diversity in the community is very worrying. If you're ever aiming to grow the community / do outreach / etc, please consider aiming efforts at ways to bring in non-white / non-male / non-"professional" folks.

Less opinionated tools and more generic tools please. In-built support for metaprogramming or ppx is valuable. The big blocker towards using ppx is the tooling as well as the documentation is not complete. Moreover, tooling such as dune makes breaking changes often based on the needs and desires of a few developers, whereas stability and long term support is more valuable for enterprise use cases. That is where OMake shines because although it is more complex to use, it is very stable and allows the users to have their own build system as they like

I didn't know OCaml before I was hired and this is pretty normal for our new hires. I'm satisfied with OCaml's overall ability to fit a wide range of projects that I work on except for fast research prototyping (Python is nicer) and some performance-sensitive applications (using C libraries / just writing in C is better). Complaints with the language are mostly of the form "please write better documentation and expect-tests and when are we getting the Eclipse debugger??"

the documentation tools could do a lot just by copying what other languages have done (rust, python, haskell), or learning from others' mistakes (how haskell docs treat typeclasses, how rust docs treat traits). the amount of different tools you have to install to get a buildchain going is an active impediment to newcomers, imho

I love OCaml but it sometimes feel like it's stagnating

I spent most of my career as a Java developer, and in comparison OCaml's editor/IDE tooling is a bit lacking. I have (was given) emacs with a working tuareg/merlin set up, which works well enough, but there's still a gap between that and the full IDE experience. A concrete example: I don't think there's a way to do "Show call hierarchy", a feature of Java IDEs since ~forever. (I couldn't find that functionality through google and colleagues don't think it exists; I might be wrong.)

Once, there was a side project for supporting sound overloading in OCaml (G'Caml I think). Has this future feature been completely discarded?

I chose typeclass above, but really all I mean is: no functors to use maps.

I think easier ppx tools/code generation tools sound nice (or better tutorials & docs). I would like to see a better build tool at my company, maybe Dune, until then my gripes with slow builds may be unfounded. I definitely don't use OCaml at home because it feels like the barrier to a nice (read: acceptable) setup is fairly high. I prefer python which works fine but has annoying runtime bugs more often. I prefer fast single threaded to multi-threaded for performance sensitive operations. Maybe encourage multi-process libraries instead of building multi-core support?

Great to see many improvements in terms of tooling and building in the past two years. Crucial missing things in my opinion are: - Standard library is lacking. - Some aspects of the syntax is confusing to people. Although superficial this is what people see when considering OCaml. F#'s light syntax works better in practice. - Simple things like pretty printing values of your custom data type is way too difficult and non-standardized - Integration with editors, tooling and windows support is crucial for more main stream adoption.

Still new to OCaml but a great language and community.

Again, the main technical problem is a bad design decisions on a high level. The lack of good old UNIX-way, minimization of dependencies, preventing the mixture of low-level and high-level code and etc.

Windows

I would also be interested in distinguishing "pure" OCaml code with "impure" OCaml code, kind of like Rust's unsafe tags, or Haskell's IO wrapper; maybe this is the direction that "effect system" would be going in.

I would really love to see strong documentation of the compiler itself. I want to get acquainted with the guts of the compiler, but I have found it quite hard to locate resources for this.

It would be nice to have tighter integration between the build system and the package manager. People who have tried Rust generally love cargo. We should strive to provide the same level of developer experience.

it badly needs debugger

I love OCaml, but the biggest thing that gives me pause when choosing OCaml over another functional language such as Haskell is it's poor support for type-level programming. It would be great if OCaml could spend some time on improving type error messages so that it could add more robust, better supported type-level programming features. If the module type system is expanded some, much of the type-level programming can be pushed into the interactions between the module level type system and the value level type system. This would make the language much more usable in larger projects where you want to enforce restrictions on a team of developer operating on disparate parts of the system. Software architecture as types and functors could be a powerful paradigm.

It would be nice to have some support for floating point vector instructions

Please focus more on Windows and Android/iOS platforms support since those are a biggest chunk of modern software development targets.

Thanks for running the survey!

Beware of the "Not Invented Here" syndrome (the opam package manager is a good illustration).

The evolution of OCaml and its ecosystem seems to be in a good way IMO. Good job and I hope it will continue in the future

OCaml need more promotion, it's not very well known language. Other languages are more popular choices, because are more popular.

One thing missing is Ocaml debugger integration with vs code. The current plugin is out of date.

thank you for caring

I like OCaml but am disappointed by having to back to mend working code written a few years ago

I like OCaml but it is extremely hard to go beyond the level of a "toy language" due to all the problems with the ecosystem and documentation.

I've used OCaml since 2001 or so, both in work situations and in my personal projects. I am saddened that, in the last few years, even as the number of people and companies adopting OCaml has grown siginificantly (it hasn't been difficult to find opportunities to do commercial work in OCaml, especially with the rise of Reason), my taste for OCaml and the direction of the language, tools, and ecosystem has shrunk. I used to view it as a rock-solid tool, simple in comparison to other functional languages like Haskell, that struck a perfect balance in terms of features versus complexity. Somewhere in the last decade, I was overjoyed that OCaml seemed to be rapidly improving; the language and tooling were actively being worked on and initially some real pain points were addressed. But, especially as I pushed to adopt OCaml internally at companies where I worked, I noticed all this change was also changing the language I liked. New libraries were often inscrutible, between LWT-all-the-things, modern Haskell-style custom operators everywhere, and heavy PPX use. New build tools guaranteed new hassles and obscure behavior that took digging through their source to uncover. I wish OCaml a bright future but I am saddened that perhaps success never comes without mediocrity.

There are four questions that I hear get when I talk to people about OCaml from average developers: where is the web API building library (like flask)? Why is it not multicore? Where is the documentation? From people who know Scala/Haskell: OCaml is unusable because it does not have higher kinded types (even if I point out that you can do the same thing with the module system)? While I have no problems with any of these things, having the answers to these four questions would be very helpful when introducing OCaml to people. If I were to prioritise any of these questions, the question about documentation. I am OK reading function signatures and figuring it out but most developers look at a list of function signatures and baulk. OCaml does not have a culture of documentation and I think it would help the community if it did.

I learned OCaml on the job and still think it's a great and versatile language. Nonetheless, compared to other (newer) languages, the community has remained fairly small and so basically, you have to find out things on your own. Also the lack of multi-core support and decent developer tools (although vscode with the reasonml plugin has changed this somewhat) are IMO the biggest hurdle to start using OCaml in an industrial environment.

It's diificult to choose only one burning desire! For the 2nd place, I'd put « a builtin "deriving" mechanism » and « a build tool »

For my work in industry, I find two main barriers. First, a knee-jerk reaction that an "other" language will be difficult to support. Second, there are some standard build and release tools that don't work well with Ocaml. So there is friction there – e.g., I have to get an executive to approve releasing things built with Ocaml as a one-off each time. For my technical work in general, the OCaml ecosystem has improved *immensely* in the past 10 years. As mentioned above, I rely heavily on the Jane Street Libraries. That mostly works well, but the web-based documentation is super hard to navigate. Often I have to click through N pages to find what I'm looking for. Also, Jane Street is not shy about making breaking changes, which can make it slightly painful when you upgrade to a new library. Of course, that's not an easy problem to solve, and the type system certainly helps. But I do sometimes wish the standard library was more full featured, or Core/Base was more stable.

Thank you for the initiative, and good luck :D

I love it, but it could be a pain in the ass

A mundane ’new’ feature I would like would be very good syntax errors from the compiler (or merlin). I don’t personally need it as I have enough experience . But I noticed while seeing beginners and when learning a new language myself that this is a very powerful tool. This removes the syntax as a barrier to entry into a language. You can play with the language without knowing the syntax (which in the case of OCaml is unusual for most ’foreigners’) and dive deep into learning what you might enjoy from the language in a mater of minutes without having to take time looking at a syntax reference.

OCaml is not well known, therefore it might be difficult to deliver software to others, e.g. in same company. Thank you for the survey!

thanks for making such a survey!

OCaml is hands-down my favorite language. I mostly use the ReasonML syntax, but I would use OCaml’s type system in just about any syntax. I really wish the community was larger, and a bit more industry-focused. Conferences, YouTube videos, tutorials, books, blogs, etc. would go a long way in spreading the word about the joy of programming in OCaml.

Opam is too different from Cargo, Ruby gems, NPM. This makes ocaml hard to use.

OCaml is excellent. I am sure the tooling and infrastructure will continue to mature, provided that newcomers adopt the language. One hope I have is that OCaml’s runtime memory representation will be made more performant sometime in the distant future, so that we can use it for stricter real-time applications.

I vacillated heavily between a surface syntax redesign and a multicore runtime. I think a syntax is more pressing, but this opens up a massive can of worms. How, exactly, should OCAML read and write? And this is a question I don’t have an answer to, today. You probably couldn’t switch to something highly consistent, like a lisp-like syntax, without massive upheaval. (Although there IS precedent for user-definable syntax, in the form of ppx, which might be (I haven’t looked into it deeply) expandable into a more general macro system) That leaves... Since reason has already cornered the C-like syntax market, I’de probably look at things like python, and maybe ruby, for inspiration. The idea would be to clean up warts and edge cases, with a focus on making interactions between different syntactic components consistent, rather than trying to make something shiny and modern.

Doing a survey is a great idea!

The low adoption rate by industry users is hands down the biggest obstacle for growing the OCaml community. If more employers were offering opportunities in this field, more developers would want to enter or stay in it.

Job opportunities are quite few!

Thank you!

Need an official debugger that doesn’t suck

Easily my favourite language, love how simple and clean it is, but it is missing maturity and for my organization to consider it we’d need multicore and good abstractions for parallelism. We currently use Scala.

OCaml is changing a lot. Once we have multicore+algebraic effects+modular implicits, I think it will be time for the OCaml core maintainers, Jane Street, and other stakeholders to come together and build a modernized and UNIFIED standard library that utilizes all these new features.

Mathematica (Wolfram) like program using OCaml would be great.

Hi!

Using the ppx infrastructure is a pain in the ass! camlp(4/5) with all its warts is more powerful and less of a maintenance headache (even if you are using dune which doesn’t really support it). I know this has been discussed on the forum but some movement on this would be wonderful. A stable user facing AST would be great! All the changes over the years with regards to language extension facilities in the language strikes of academics piss marking. Not good for the community!

For larger applications, it gets harder to reason about performance, and harder to measure it will the available tools. We still have to jump through hoops to make use of multiple processors efficiently.

Lwt is really important third party library for OCaml(I think so) But the document is not enough for me to understand. I could not understand "What is Asnyc(or non blocking?)" and how can i handle that with Lwt. For example, I created a RESTful Service with Opium framework. This works fine, but could not correct handle large uploaded file.(OCaml process blocks other http access when large data incomming and processing in Controller) Usually uses a thread for that. but OCaml could not use thread. therefore exists Lwt. but Lwt is difficult to understand... OCaml needs documents and sample codes for server side applicat

Making a docs.rs equivalent would do more to improve ocaml development than anything else (such as multicore)

I feel the planned "docs.ocaml.org" providing a consistent interface to documentation for all OPAM package is important.

BTW, I'd also like "an effect system", "effect handlers", "namespaces", and "modular implicits" but how could I not choose "multicore runtime" right now?

I like ReasonML, ReasonReact, ReScript.....I started learning OCaml and I would love to get my first job with it!

Keep up the great work. Ocaml deserves it;

For the last question, another answer I would have given is not there. I think a big pain point that prevents OCaml adoption is "Lack of marketing".

OCaml lacks a hollistic solution / golden path to be more used in applications. Removing friction to newcomers and improving the ecosystem in a holistic manner. So applications can actually be built in OCaml. And please, never mix politics and technology. That is really horrible. Some projects have done it and it does not go well. ‘Code of Conduct‘ for example.

You're too fragmented. There's Jane St. libs and then there's everything else... Could we have one a single Async lib?

Thank you for doing this survey! I hope you receive a lot of useful feedback. I'm excited for what the OCSF has in store for the future.

One of the things I miss the most is an easier way of embedding ocaml programs in not very common places, like micro controllers.

Please, please we need Cargo for OCaml

Documentation should include lot of examples (like matlab do)

Recent improvement on OCaml itself and its ecosystem is amazing! I really appreciate that.

You're all doing great!

Better gdb support would be nice; that seems to have stalled in lieu of other things. But really: API documentation, it's so bad. :(

Multicore initiative should be the highest priority.

I sometimes come across people complaining about how difficult it would be to hire OCaml developers or train developers to use OCaml, but I get the impression they haven't actually tried.

Converge to one state-of-the-art standard library like in Rust

I love ML and OCaml is a blessing to have in this language family. Let's make MLs mainstream!

Don't invest too many resources in front-end web development, there's too much churn in that and no way to get traction. Stick to OCaml's strengths.

Please merge modular implicits the official OCaml compiler

I chose "too hard to learn" only because there is so much choice in competing libraries/build tools/etc and not enough guidance on "here's a set of libraries and tools that will work for most people". https://ocamlverse.github.io/ is a good step in this direction and is worth keeping an eye on.

It is a great language, but there should be a shorter way in for beginners. Like tutorials (that are updated).

Would be nice to have more support for macros in the language - ppxlib tends to be a bit of a heavyweight solution, I'd ideally like something with the ease of use of lisp macros.

if ocaml is easy everything else becomes easy(in the survery). my choice for language feature came to a draw with "a complete redesign of the surface syntax". I ticked "typelcasses" only because I think I need them to write shorter code in the short term (my concrete priorities). I have been learning OCaml in the last year. The module system is a barrier. It's a second syntax = more cognitive load, more distance to first real world project. I don't use the module system just because I have to learn yet another thing. It cleaves the beauty of ML. I think I can observe the syntax additions across the years. History unfolds in the manual and in bare code. The surface syntax is therefore incoherent, clumsy, unclean (and this ripples mecanically in code onwards, polluting the ecosystem = syntaxic entropy/complexity). Accessing any new features is learning yet another weird syntax (ppx, let, module). Maybe it's time to integrate, flatten, abstract everything nicely (modular implicits is rumored to be on stall due to this, idk, but technical debt maybe). Sometimes I can see the patchwork (even the O prepended to Caml stands for this point). Caml seems more like OPQRCamL. Anecdotically, I heard Leroy teach at College de France something about type universes in a grand summary of "OCaml" research. I don't know if it captures the module system. OCaml ressembles a bit natural languages after 10 generations : new comers don't know why it is alambicated anymore. In a nutshell I am not simply learning modern academic ML but its history. I can't agree more with the value of learning how it was made but it is important to have a recently cast language for use (I am writing to you with the point of view of a student). Maybe the problem has to do with backward compatibility. Don't you think it's funny ? All mathematics disregards backward compatibility except computer science. Is it because of libraries ? Maybe having a glue to backread anciant caml is fine. I compiled caml source code dating back from the year 2000 thanks to the github time machine (all interpreters being available). Didn't change a single line of code. If only I was an expert *already* I 'd do a tool or extension to address per file interpreting with an *interpreter set* of ocamls. In all mathematics, computer science remains the exception that sticks to backward compatiblity. This is maybe because of the link between academics and the industry in the case of ocaml (I have not thought of these questions yet TBH and I have a narrow perspective on this :/). As you can guess, I am a hard believer of so-called "breaking change". Simply put, backward compatibility is just continuous change. That intrinsically bares a limit to change and is plain incompatible with development (evolution). On the upside I chose ML because with functions (I think of core ML) I can theoretically glue everything. Be it anciant Golang network libraries or 1990 ocaml. When time comes I will dig this question continuing from the idea of Js_of_ocaml. Plug and play at lower , more stable levels. Beginning with ocaml to ocaml interfacing so we can remove the backward compatibility brakes. Last but not least, I don't think OCaml is a language. I think it's a group of individuals doing research. To me, OCaml is a representation of the cumulative work. This is why I see OCaml as an academic course rather than a PL. Hence my remarks on tidying up knowledge in a clear distilled form for new students who will build on top of it. Think of geometry, algebra and cousins. Finally, I must admit I am personnally frustrated of learning so slowly. Otherwise I would contribute already to this endeavour. I am happy I could participate in the debate though. Cheers :) see https://en.wikipedia.org/wiki/Turkish_alphabet regarding breaking change.

I think outreach to minority and underrepresented devs and learners could be hugely valuable (following rust in this, but perhaps even more regressive!) Following Lawvere's insight about the role of foundations in education, this will push to to improve the core. Moreover, there is a huge untapped audience and user base in those who are not sufficiently welcomed, encouraged and empowered by the often toxic tech mentality

is there any future for ocaml

Thank you!

Take time to learn. Not many resources for deep concept. hard to adopt because people good in other languages do not want to make the effort

OCaml great , but Reason more friendly.

Keep up doing what you are doing !!!

I’d love to see PPX support manipulating typed ASTs!

A.6 How should we improve this survey in the future?

Do it regularly, biannually if possible

Good survey!

Some questions are in the second person while others are in the first person, and that sounds weird.

not hosting it on a big-corporation platform that doesn't respect privacy

I have no idea what an Ocaml Project is.

Why is 4.04 not included in the choice? I have been using 4.04 for quite a long time.

Come back every 3 years

What if ive been using OCaml for 1.5 years?

Generally better survey hygiene: - Alphabetical sorting of multiple-choice questions to avoid confirmation bias by frequency; - Sound ranges (not disjoint, e.g. "less than 1 year" & "2-5 years"; or overlapping, e.g. "2-3" & "3-10"); - Consistent grammar / use of punctuation etc.

I think the most important thing about this sort of surveys is doing them regularly, and communicating the results back to the community! ;-)

All good.

Thanks for doing it!!

"If one piece of the ecosystem could magically be made state-of-the-art, I would ask for:" - allow to order the items in terms of priority Ask for what precise problems people had as beginners and couldn't find solutions for - this will help you figure out what is particularly bad

Specify whether "OCaml X" refers to public X, or any X the respondent has. I was uncertain how to answer some questions because I have generally positive experience with our proprietary stuff and no experience or poor experiences with the public stuff. I'm glad you included diversity as a possible goal. Include "I don't know" as an option in more questions. Include better compile-time evaluation (e.g., macros) in the "one new language feature" section.

it's pretty good already :)

Having multichoice for burning desire.

Let us pick more than one thing to improve!

once you have a better idea of what libraries people think are missing, maybe add a question on this to have a better poll of this issue?

It seems fine to me!

This was great, quite a fun survey! Thanks for putting this together.

Publicize it more actively; maybe on Twitter :) Another area of interest potentially worth exploring is what are the projects people would wish to contribute to and what blockers exist if any.

questions about meta-ocaml

I think it's good

It was good, thank you.

A couple questions that are single option should be multiple choice.

You might want to add "not applicable" as an answer to a few questions.

If next could be done without google, it would be better...

Perhaps you could benefit from knowing the region in which we are answering to learn region-based specificities and needs?

More open questions about language developments

Add country question. So that you can know which country has more ocaml developers.

Ended up feeling pretty commerical focused, and I'm a wee academic.

N/A.

Maybe get into more details about how the tools are used, and what tooling people think is missing. Also, asking about what people are ready to do in order to make their libraries/documentation/tools more useful and accessible. Mostly, have this survey as often as reasonable. We're getting to a point where a thread on the caml-list can't work anymore and maintaining community communication is crucial. Thank you!

Give more examples of popular third-party libraries from other languages and ask how useful it would be to see analogues of these in OCaml.

I feel this survey might be too biased :) but it's just a feeling, I don't have any proof of this!

First question did not have alternative for 1-2 years, only less than 1 and 2-5.

The question about "improving diversity" is weird, if not shocking. Obviously, if the ocaml community lacks diversity, few people are likely to choose this item. Then you can say, "we are good on diversity"... my feeling is, you are not. Diversity should not be a user's feeling. It should be measured. And as long as you don't have roughly 50% women and a reasonable amount of people of color, you are NOT good.

:shrug:

I am liking it as it is

Tiny nitpicks, but there were a couple of questions in the first section where it wasn't clear how to answer if you only use OCaml in your free time. I hadn't realised I could leave those questions blank!

Spend more forms polling about teaching. For years, OCaml has been taught in French schools and universities, and I think it's unfortunately and paradoxically decreasing. The fact that this very survey does not consider teaching that much shows a shift in OCaml towards developers, which is good, but at the expense of teaching.

Make them yearly, at least.

Didn't mention Windows!

N/A – seems good.

Some of the questions in page 2 and 3 were missing a "Not Applicable" option

Ask more questions where users can compare to other languages :)

Ask about debugging needs

Include formal verification explicitly as a "testing" option

The "If I was granted one new language feature today" question needs better wording/explanation. I understand what "multicore runtime" and "redesign the surface syntax" are but I can't guess what the other items might be or their benefits. They need explanation, perhaps in some detail. And the list doesn't include things I really want, such as a much better debugger. I think I'd look for a more open process for prioritizing improvements: First, solicit open-ended discussion on discuss, then have someone boil that down into a list that people can vote on. Perhaps should be separate from the survey. It would also be valuable for someone to review all the enhancement requests to identify relevant topics and ideas–though that takes some effort. A voting system may be useful though that would be a bit of work.

No comment, it's a great idea though, thanks for running it!

First question doesn't have a "1-2 years" option. Some questions doesn't have the answer I have in mind but also doesn't have an "other" option (for example, what to make state-of-the-art).

In some universities, OCaml is used in instruction. In Tsukuba University, for example, OCaml is used in a lab courses for 3-year students (write an interpreter, type-checker, inferencer, etc). In our lab and some other labs, there are similar lab projects for our students. It would be good if the survey contained some questions of using OCaml in education, in lab courses, PL language courses, etc.

less talk, more action :)

nothing good job

In one of the questions there was no option for "BuckleScript with OCaml" (which I have used a little)

Add an option for whatever merlin is to the "make one thing state-of-the-art" question

Improve OCaml communication so it is no longer the weird programming language people used (in an imperative way) at the university.

Use rank choice rather than “choose one” for the burning desires question.

It is really good.

For the question "For how long have you been using OCaml?" there was no option for "1 year", and there were two options for both "5 years" and "10 years".

Missing option for 1-2 years experience, make it clearer what the "What would you change" questions on page 8 correspond to.

I didn't understand what "dynamically linked vs statically linked binary" meant in some question. ocaml libraries are statically linked unless you do and do something custom. Was that talking C libraries? For the question of "what's the oldest compiler version I care to support", the answer is not one version, it's "it depends on the project".

This was great, thank you.

Perhaps include a section on usage. Frontend, backend, machine learning etc.

More options in between "neutral" and "agree"/"disagree"; there were some questions I felt like I was "slightly agree" on, but "agree" felt a little too strong

Good survey, no feedback here.

Add more questions about ecosystem (libraries).

Don't use Google forms, please

It's ok as it is now

Always add an open point at the end of the selection choices

Not now, it is good for now I think

There were several questions where I had to put "None" in myself. Having more "None" or "I don't know" options would help.

Nothing to suggest. Great idea!

I think this survey is very important and you are in right path.

It's ok, keep up the good work !

by making it on a regular basis

For the "years of using OCaml" question, there should be an option for "1-2 years"

The survey itself is quite well designed. Nicely done.

Ask about demographics.

The survey leaves little place for "different" input than what is being discussed in the OCaml community anyways.

n/a - it was good.

Do it each month. Frequency. Repit somthing until we adopt uses and customs

This is really great action that you hear developers opinion. But a lot of projects(languages, framework, OS etc) did same and i did NOT yet see the result. Only hearing. But nothing changed. I hope if you hear developer's opinion continual and really use the opinions.

Survey the quality of editor integrations (LSP features, etc).

It's great. Keep doing it, and make sure it is representative of full range of target audience. Need lots of responses or the results may be misleading.

Differentiate between pure developer and hiring manager. If corporate wants to move away from OCaml, there's almost no stopping them.

one of the single choice questions had checkboxes.

thank you for listening.

I don't know

I wasn't able to choose Ruby from the language list, despite that it's the one I use on the job (full time Ruby/Rails development for a web service).

I don't know. But this offloads many discuss.ocaml.org discussions, frustrations, about dev schedules. It surely lifts the mood :) Nice thing this survey is made public.

Thank you! <3

its good

I don(know