OCaml Users Survey 2022
Summary of results

OCaml Software Foundation
December 26, 2022

1 Introduction

The second iteration of the OCaml Users Survey was conducted from March 1st 2022 to March 11th 2022. It was devised by the OCSF using the 2020 Survey as a template, and improving it using the feedback from the first iteration. The Executive Committee is grateful to Claude Rubinson who helped further improve the phrasing and framing of several questions. We also thank the respondents who made constructive remarks on or offline, these will be integrated in future iterations.

The survey attracted 280 answers, a sharp decline w.r.t to the first iteration (745 answers). It is not clear why that is the case. The survey was announced on discuss, on the OCaml mailing-list, on reddit and a few other channels. The statistics on the discuss post seem to indicate that only 104 people followed the link through this post. While we do not have a clear explanation, several people mentioned their reluctance to use a Google owned service (in that case Google Forms). Other factors may include the global geopolitical situation, the "lack of novelty" (since the survey is similar to the previous one). It also seems from the answers that in the previous survey, a significant part of the respondents were Reason users, which now seem to be have drifted away from the OCaml community.

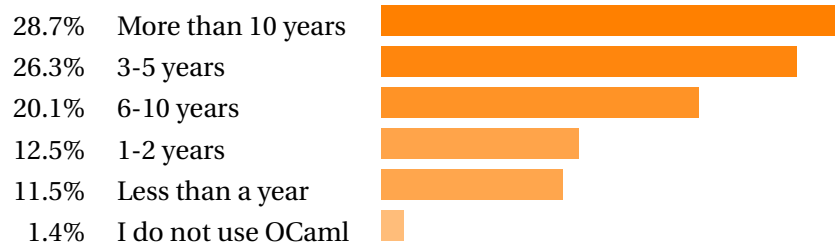
In Section 2 we summarize and briefly comment on each question. Section 3 draws preliminary conclusions.

2 Survey results

Convention for each question we give the number of respondents in parentheses. For questions allowing multiple answers, we also give the most frequent pairings when they are meaningful, i.e. when they seem to indicate a correlation between two answers. In such results, a line of the form "x% A/B" means "for all answers consisting of at least two choices, combination A/B occurred x% of the time" and the last line of the table give the percentage of respondents with multiple answers.

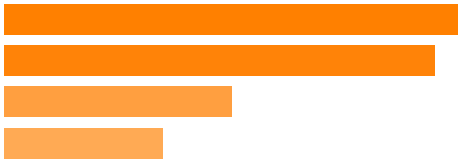
2.1 OCaml Usage

Q1: How long have you been using OCaml (279)



Almost 50% of the respondents are seasoned OCaml programmers, having used the language for more than 6 years, while 33.7% have used it for less than two years.

Q2: How do you rate your OCaml proficiency (280)



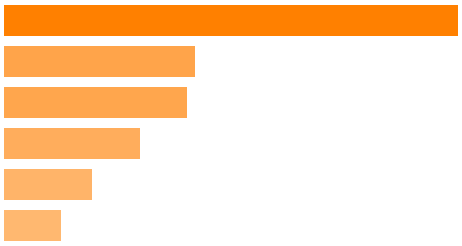
35.7%	Advanced
33.9%	Intermediate
17.9%	Expert
12.5%	Beginner

Interestingly, there are far less users considering themselves beginners than there are recent users of the language.

Q3: In which context do you use OCaml (279)

54.5%	As a hobbyist
46.6%	For work, in industry
30.5%	For work, in a research job
11.5%	As a student

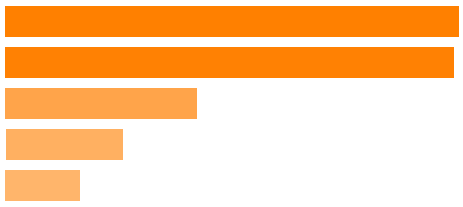
Other answers (with 4 respondents or less) include : *teaching, open source contribution, internship, coq contributor/volunteer*.



38.8%	For work, in industry/as a hobbyist
16.3%	As a hobbyist/for work, in a research job
15.6%	For work, in industry/for work, in a research job
11.6%	As a student/as a hobbyist
7.5%	As a student/for work, in a research job
4.8%	For work, in industry/as a student

prop.	*40.9%*

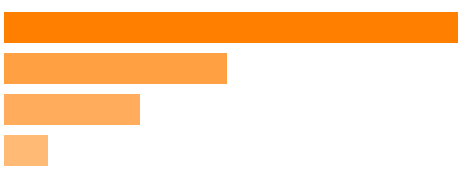
Q4: OCaml usage at your workplace (275)



35.3%	Most (> 50%) software projects use OCaml
34.9%	I am –or suspect that I am– the only OCaml user
14.9%	Some (> 5%) software projects use OCaml
9.1%	Many (> 20%) software projects use OCaml
5.8%	I don't know

OCaml appears to be an all or nothing solution! One third of the respondents seem to be working in an OCaml centric environment while for an other third, they appear to be the only OCaml user.

Q5: OCaml trend at your workplace (270)

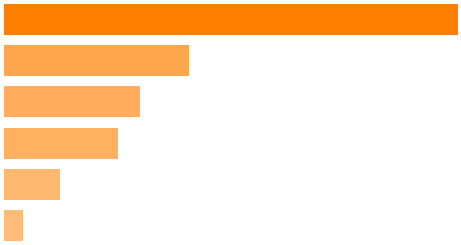


54.4%	OCaml usage is stable
26.7%	Unsure/I don't know
16.3%	OCaml usage is increasing
5.2%	OCaml usage is decreasing

These results do not vary much from the previous survey (taking into account it did not have "Unsure" choice).

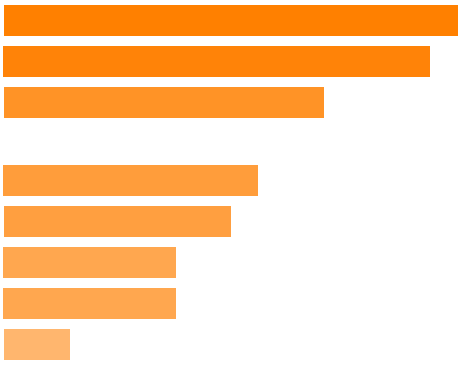
Q6: How did you learn OCaml (279)

70.6%	Self-taught
28.7%	University, undergraduate level
21.1%	At work
17.6%	University, graduate level
8.6%	Via an online course
2.9%	In high school



A mistake in this question allowed the respondents to select multiple answers. Interestingly, amongst the 70.6% of respondents who answered "self-taught", 41.9% answered only that choice, while the rest answered self-taught together with other choices.

20.0%	At work/self-taught
18.8%	Self-taught/university, undergraduate level
14.1%	University, undergraduate level/university, graduate level
11.2%	Self-taught/via an online course
10.0%	At work/university, undergraduate level
7.6%	At work/university, graduate level
7.6%	Self-taught/university, graduate level
2.9%	At work/via an online course
prop.	*38.6%*

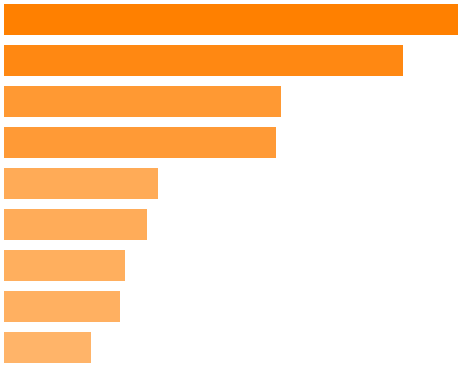


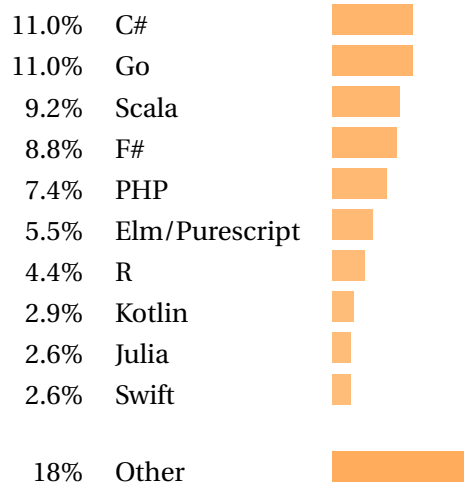
Q7: If you learned in school or University, which one (optional, 101)

Of the 101 respondents to this optional question, 64 learned OCaml in France (at University or in "*classe préparatoire*"). Other countries include Portugal, Belgium, Germany, the Netherlands, the UK, the USA, Canada, Israel and South Korea. See Appendix A.1 for the complete list of answers.

Q8: Which of these other programming languages are you fluent in (272)

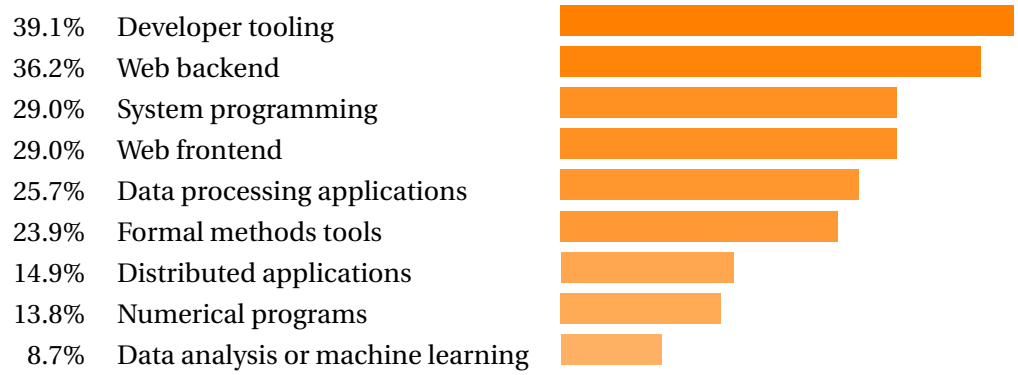
62.1%	C
54.5%	Python
37.9%	Java
37.1%	Javascript/Typescript
21.0%	Scheme or Lisp dialect
19.5%	Haskell
16.5%	Rust
15.8%	Coq, Lean, Agda or Idris
11.8%	Erlang or Elixir





It seems OCaml programmers follow the global trend, many of them being fluent in C, Python, Java and JavaScript. It is also not unexpected to to see other functional languages at 20% (Scheme, Haskell). And yes, we forgot Perl in the possible choices. Although a majority of respondents (85%) listed at least two languages, no combination strikes as particularly representative, almost any possible pairing occurs in the result, with the top ones being C/C++ and C/Java with less than 3.4%.

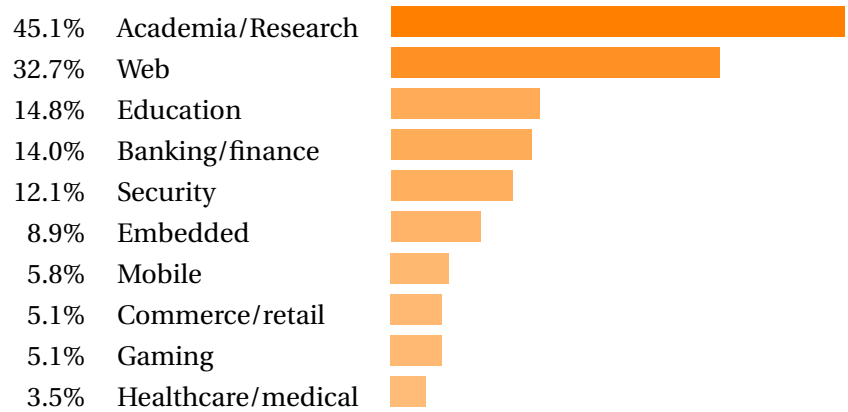
Which type of software have you or do you currently develop with OCaml? (276)



Other: Code generation for high energy physics, Monte Carlo Event generators, Conceptual Graphs,CAD (computer assisted design), prototype, video rendering, Small everyday utilities (replacement for .sh files), Toy programs (demos), Prototyping Simulations for Games/Visual Effects before implementing in C++ / Rust, WebVR / OpenXR,music production, Fun stuff & pet projects, proof assistants, crypto PoC, scripts, Cross-platform desktop GUI (text editor), trading software

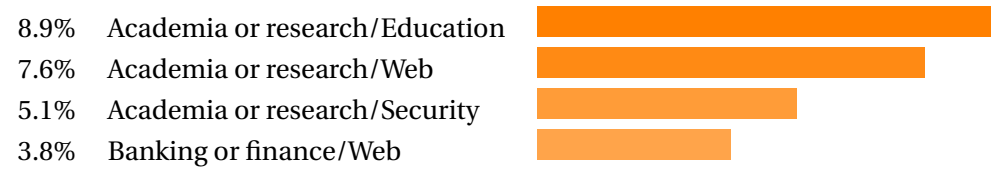
Again all possible pairing are well distributed (the top-most one being tied at 5.7% : "Web backend/Web frontend" and "Developer tooling/Programming language implementations").

In which domain is the software you develop used? (257)



Other: *personal ergonomics, Virtualization, sport , publishing, government, Personnal small tools and apps, Entertainment & Arts, Industrial data acquisition, Systems maintenance, Contract management, Networking, Open source, General CS, I contribute to Lwt which is used everywhere?, Where ever OCaml is used, It's tooling, mostly everywhere, Iinfra, tooling, Messaging, telecommunications, music production, Avionics, Automotive, Railway, Industry, telecom, hobby, Any, Not used, it is mostly student projects, system utilities and tooling, industry, telecom, Industrial automation, personal use, hobby project, Agriculture*

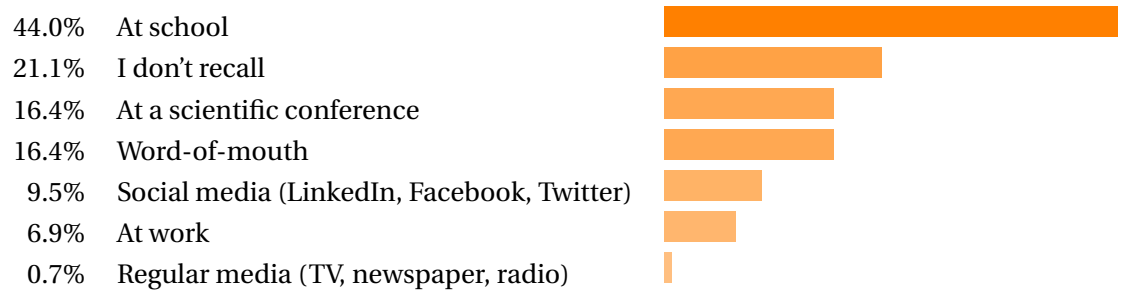
OCaml seems still mainly used in an academic or R&D setting. As with the previous survey, the Web industry and financial industry are well represented. We surmise that the relatively high score of Education is due to respondents developping software as teaching material for courses such as functional programming, compilers or formal methods (rather than developping software used in the education industry).



prop. *43.6%*

The top-most pairings are these followed by a long tail of other possible combinations.

Q9: How did you first hear about OCaml? (275)



School, and particularly higher education, still is by far the first mean of diffusion of the OCaml language.

Q10: What is a pain point when learning the OCaml language ? (183)

Free form question. See the raw answers in Appendix A.2.

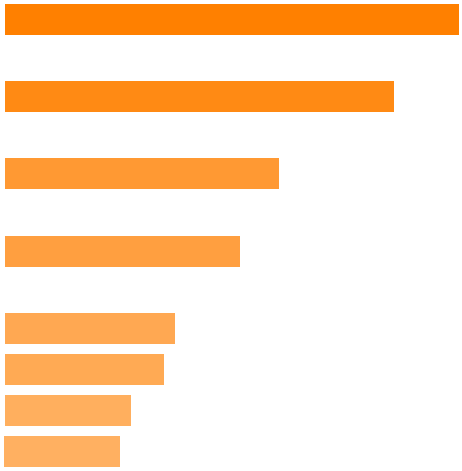
2.2 The OCaml Software Foundation

Q11: Did you know about the OCaml Software Foundation (OCSF) before this survey? (280)

68.2%	Yes
31.8%	No

An improvement with respect to the previous survey (51.6% yes, 46.4% no).

Q12: What topics or actions would you like the OCSF to work on? (272)

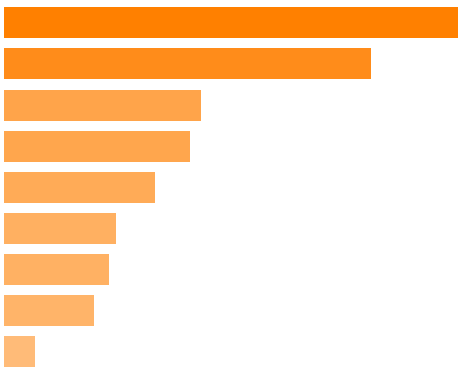


76.8%	Support the development and evolution of the wider OCaml library and ecosystem
65.8%	Support the development of the OCaml language, compiler and core tools
46.3%	Promote the language in the Industrial world (through conferences and events sponsorship)
39.7%	Encourage and support OCaml courses for students (Universities, schools, ...)
28.7%	Animate the OCaml community
26.8%	Fund better support across OSes and architectures
21.3%	Increase the diversity of the OCaml community
19.5%	Encourage and support OCaml training for professional developpers

Other: *Help organize OCaml skill supply & demand in the global job market, Support development and maintenance of bindings to key external libraries, Better IDE experiences, uncover and accelerate fixing ecosystem gaps for industry adoption, Support it for mobile development to make it truly cross-platform, equivalent of libbacktrace in Ocaml, Improve documentation, esp. training materials. See for example Guido's Python tutorial in the Python documentation. All are worth do what you think is best, Substantial improvements to the documentation and elitist attitude which closes it off to new comers, Make the ecosystem stable and usable for professional work, Marketing, promote the language to a wider public and institution*

2.3 Community

Q13: Where do you interact with other members of the OCaml community? (269)

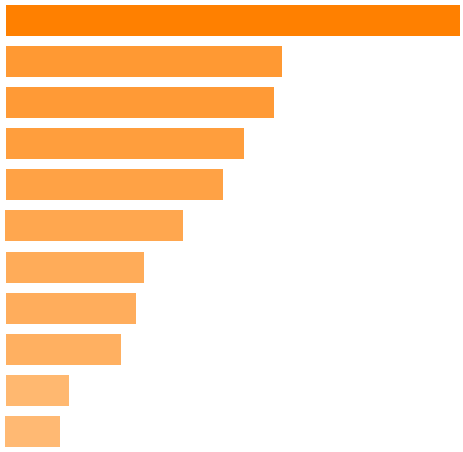


77.3%	`discuss.ocaml.org`
62.5%	GitHub
33.5%	IRC/Discord
31.6%	Caml-list
25.7%	r/ocaml
19%	Twitter
17.8%	Conferences (academic)
15.2%	Slack
5.2%	Conferences (industrial)

It seems that `discuss.ocaml.org` has become the primary mean of interaction for members of the community, closely followed by GitHub (where we assume technical discussions happen in PRs and bug reports). We forgot to include the `matrix.org` OCaml channel, it will be added in the next iteration of the survey.

The pairing follows the aggregate results above, with `discuss.ocaml.org` begin in the top-most significant pairs of answers.

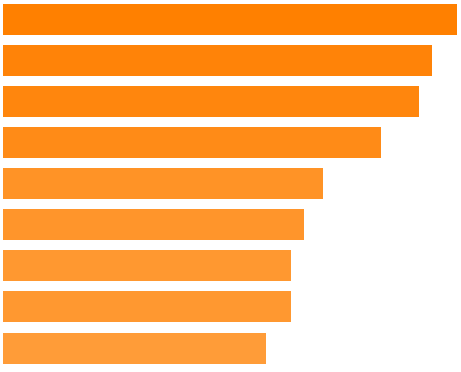
Q14: Which of the following topics would you like to see more contents about? (261)



57.9%	Best practices
35.2%	Tooling choices
34.1%	Debugging how-tos
30.3%	Library walkthrough
27.6%	Performance analysis
22.6%	Case studies
17.6%	Web development
16.5%	Beginner fundamentals
14.6%	GUIs
8%	Machine learning
6.9%	Comparison with other languages

Other: *ELI5 (non-academic) content about more theoretical pre-requisites for participating in discussions about the language, Cross-compiling, embedded, Windows,Documentation on advanced topics (universal quantifiers, variance, ppx, camlp4/5, gadts, effect hanlders,...), Scientific applications, polyglot development coexisting with golang/python/rescript/typescript, how to use ppx libraries*

More than half of the respondents would like to have more content about how to do things *the right way*^{TM}. The pairing results further confirm it:

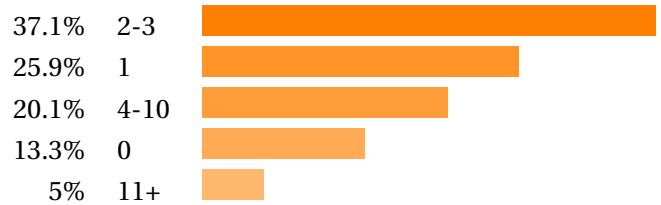


7.1%	Tooling choices/Best practices
6.7%	Best practices/Debugging how-tos
6.5%	Case studies/Best practices
5.9%	Library walkthroughs/Best practices
5.0%	Debugging how-tos/Performance analysis
4.7%	Beginner fundamentals/Best practices
4.5%	Library walkthroughs/Tooling choices
4.5%	Best practices/Performance analysis
4.1%	Tooling choices/Debugging how-tos

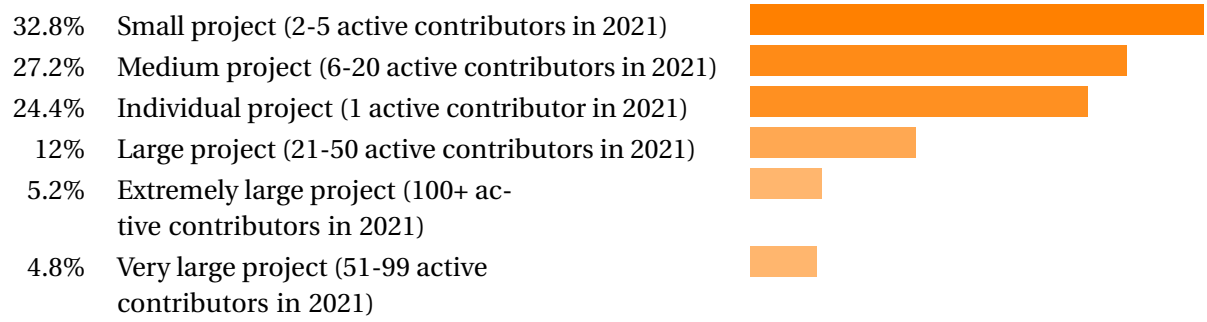
prop. *87.6%*

2.4 Projects and contributions

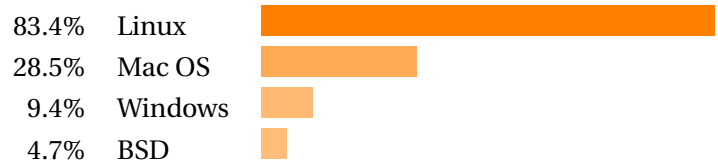
Q15: About how many OCaml Projects did you contribute to in 2021? (278)



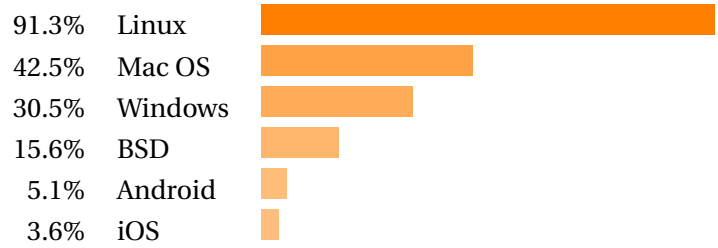
Q16: How large was the largest OCaml project you contributed to in 2021? (250)



Q17: In 2021, which platform did you use to write OCaml code? (277)



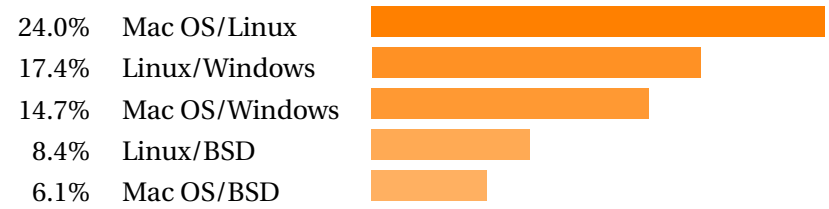
Q18: In 2021, which platform(s) did you target? (275)



Other: *Web, JavaScript, MirageOS, Solo5-hvt, browsers, unikernels, Oculus Quest 2, WSL*

The three main server/desktop OSes are still the principal targets, with Linux being almost always supported by software developed by the respondents. The option "Web browser" will be added next year as it seems to be a target in its own right (e.g. via `js_of_ocaml`). While only 9.4% of the respondents primarily use Windows to write code, 30.5% target it explicitly. A first step to improving "Windows support" as often requested, could be to improve cross compilation for Windows from Linux.

As expected, the combinations of the main three server and desktop OSes constitute top pairings.



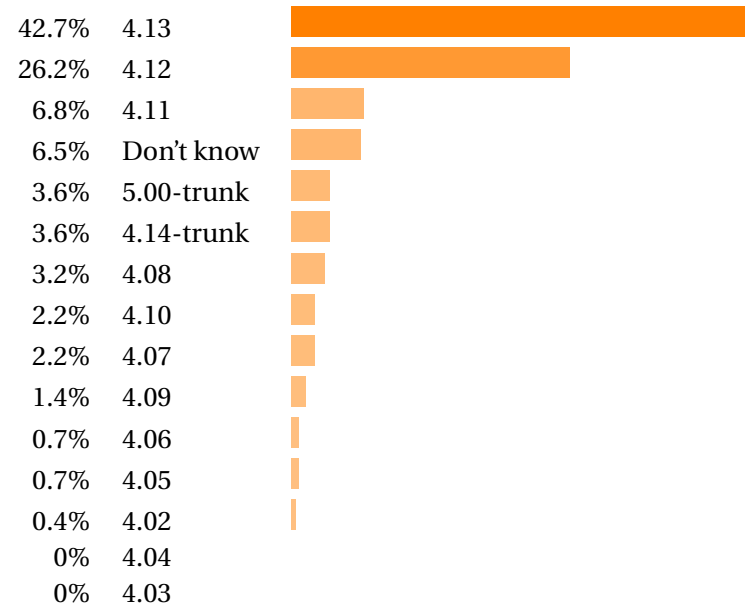
prop. *49.6%*

2.5 Compiler

Q19: Which of these language implementations are you using, as of January 1st 2022? (276)

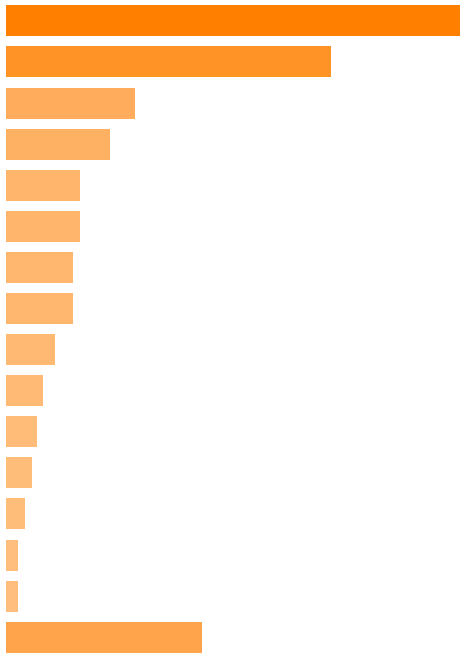
Compared to the previous survey (respectively 91.6%, 30.8%, 16% 9.3%), Reason usage seems to have decreased amongst the respondents.

Q20: Which version of the OCaml compiler do you use most, as of January 1st 2022? (279)



The overwelming majority of the respondents uses a version of the compiler less than a year old (OCaml 4.12 was released in February 2021). The use of OCaml below 4.07 amongst the respondents is insignificant.

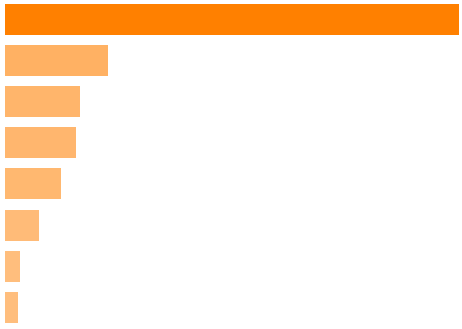
Q21: What is the oldest version that you try to support in the software you develop? (273)



27.1%	Only the latest release
19.4%	4.08
7.7%	Debian stable
6.2%	4.13
4.4%	4.12
4.4%	4.11
4%	4.10
4%	4.07
2.9%	4.05
2.2%	4.02
1.8%	4.09
1.5%	4.04
1.1%	4.06
0.7%	4.03
0.7%	The current version supported by Bucklescript
11.7%	N/A

Besides the "latest release", it seems that OCaml 4.08 is the oldest release for which compatibility is deemed important. As a reminder, OCaml 4.08 was released in 2019, and was the first release to feature the `Bool`, `Fun`, `Int`, `Option` and `Result` module as well as the `let*`, `let+` and `and+` operators.

Q22: Which installation methods do you use? (275)

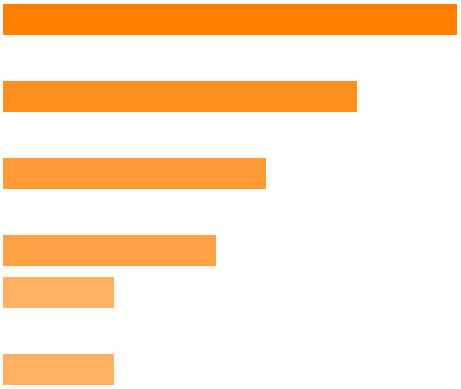


88.4%	Opam (using the public repository)
20%	Opam (using a private repository)
14.5%	Manual compilation and installation
13.8%	Distribution's package manager
10.9%	Monorepo
6.5%	esy
2.9%	npm
2.5%	Manual installation of pre-compiled binaries

Opam is the installation method of choice. We respect to the previous survey, the use of the public repository has increased (80 to 88.4 %), the use of the private repositories is stable (20%) while the use of monorepo seems to have decreased (from 14.6% to 10.9%).

When respondants use more than one method, Opam is still in the 5 top choices:

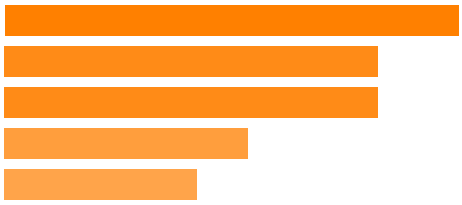
20.9%	Opam (using a private repository)/Opam (using the public repository)
16.3%	Opam (using the public repository)/Manual compilation and installation of source code
12.1%	Opam (using the public repository)/Distribution's package manager
9.8%	Opam (using the public repository)/Monorepo
5.1%	Opam (using a private repository)/Manual compilation and installation of source code
5.1%	Esy/Opam (using the public repository)



prop. *42.9%*

Q23: How long do you typically wait after an OCaml release to test your codebase with the new version? (For whatever reason : by choice, because your dependencies are blocking you, or any other reason) (249)

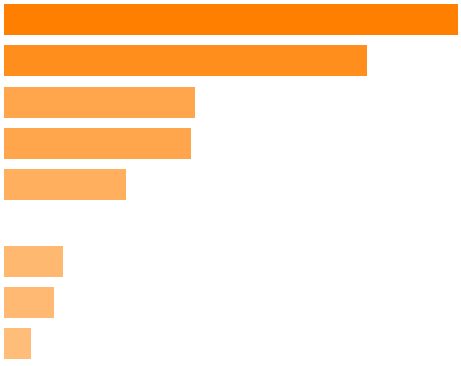
29.3%	Less than a week
24.1%	Less than a month
24.1%	Less than three months
15.7%	One release late (≈ 6 months)
12.4%	More than one release late (more than 6 months)



The vast majority of respondents test their codebase within the release cycle of the latest version (less than 6 months).

Q24: Once you test your codebase to the new OCaml version, what is painful and requires adapting your project? (228)

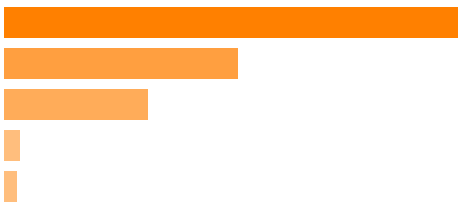
43.9%	Working around incompatible dependencies
35.1%	It's perfectly smooth each time!
18.4%	New compiler warnings
18%	Breaking changes
11.8%	Reliance on compiler internal APIs (`compiler-libs`)
5.7%	Compiler bugs
4.8%	Other
2.6%	Reliance on the Foreign Function Interface



By far, the biggest source of incompatibility seems to be related to incompatible software dependencies.

Q25: How do you feel about the compiler's every-six-month release schedule? (276)

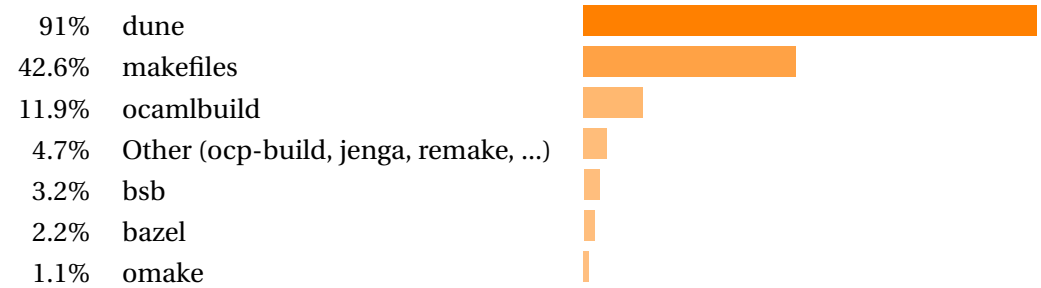
52.9%	It's just right
27.2%	I'm indifferent or unsure
16.7%	I was not aware of it
1.8%	It's too slow
1.4%	It's too fast



The vast majority of respondents has either adapted to the every-six-month release schedule, or is not impacted by it.

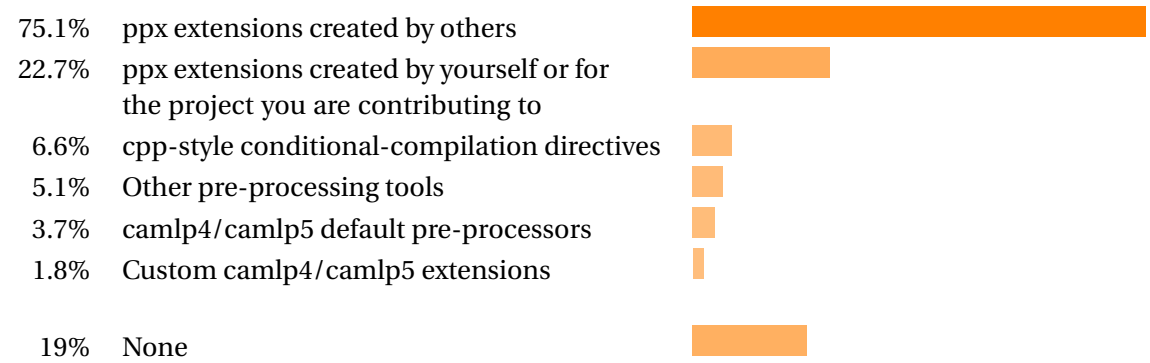
2.6 Tooling

Q26: Which build tools are you using, as of January 1st 2022? (277)



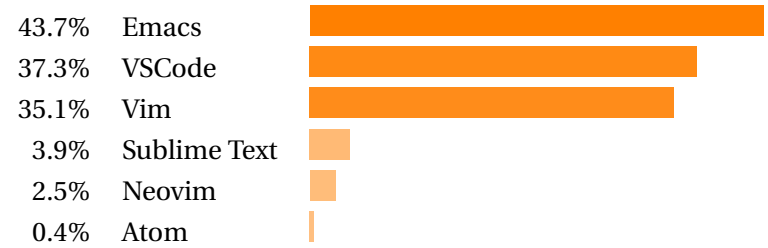
While Dune has become the de-facto standard for building OCaml projects, 42.6% of the respondents still use Make. The rest of the build tools are far less represented.

Q27: What kind of pre-processors do you use, as of January 1st 2022? (273)



The respondants seem to have widely adopted PPX style extensions, for those who use preprocessing.

Q28: Which editors do you use, as of January 1st 2022? (279)



Other: *Kakoune,Geany (GTK), Crimson Editor (Win32), Intellij, vis, mg, Spacemacs, JetBrains CLion with OCaml add-in, Notepad++, acme, IntelliJ IDEA (with ReasonML plugins),Pluma*

Interestingly, some respondents use several editors, the raw answers even indicate some using both Vim and Emacs.

29.9% VSCode/Vim
29.9% VSCode/Emacs
14.9% Vim/Emacs
5.7% VScode/Sublime Text

prop. *24.1%*

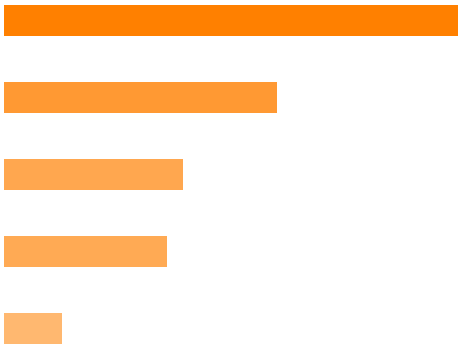
Q29: Where do you host your software sources, as of January 1st 2022? (274)

As expected, GitHub being is overwhelmingly used by the respondents. The pairing results below indicates that most often, GitHub is always in the mix even when other hosts are used.

prop. *37.2%*

This can easily be explained by the fact that most high-level projects, including the compiler itself are hosted on github and that tools such as Opam or Dune have special support for this platform.

Q30: Which tools do you use to test OCaml code, as of January 1st 2022? (245)



Other: *Cram tests,Manual,Homemade testing system, bisect_ppx, home made test scripts, I do not write tests, I just have "examples",REPL, end-to-end regression testing,Expect style with additional code for comparison, regression tests using output to text files, custom test system a bit similar to expect test but less strict on output, Alcotest, None, Custom testing scripts, our own non-regression testing tool, interface QA testing*

Q31: Which tools do you use to benchmark OCaml code, as of January 1st 2022? (178)

82%	Whole program time measurements (time, perf-stat, valgrind, gprof...)
31.5%	Microbenchmarking library (core_bench, benchmark, bench...)
11.2%	Performance-monitoring libraries (landmark...)
3.4%	Preprocessor-helped microbenchmarking (ppx_bench...)

Other: *None, Memtrace, Benchmarking library built into the project, Most tools tend to provide unstable results for my use case, emit catapult format with manual probes, memprof, Looking at the CMM, custom profiling function, Self-built tools, explicit calls to timing functions, nothing, don't understand*

The vast majority of respondents use (at least) whole program measurements. It also seems to confirm that OCaml's binaries and its runtime play rather well with standard tools such as `perf-stat` or `valgrind`.

2.7 Infrastructure

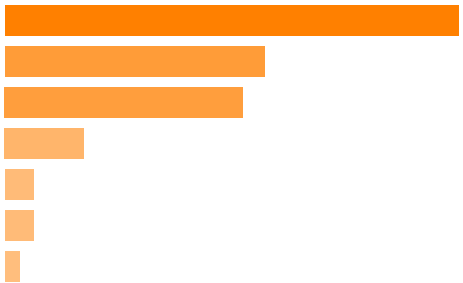
Q32: How do you deploy OCaml applications? (240)

37.9%	As static binaries
33.8%	Through the user's package manager
32.1%	As docker images
30%	Through the Web
16.3%	As dynamically-linked binaries

Other: *open source code only,None,Source users compile,singularity images, as shared libraries, Nix. Otherwise I let the user figure it out. The distribution of OCaml program is not fun right now, Internal packaging infrastructure, git clone, opam, Via opam (though that is not ideal), X, Google Play Store / Apple App Store,I write libraries, GitHub release tags, I'd like to have a *simple* way to create static binaries, I do not, opam*

For respondents who do not deploy Web applications, they seem to prefer having the dependencies handled (by the package manager, bundled in a docker image or with static binaries), rather than distributing dynamically linked binaries.

Q33: If you deploy on the web, what platform(s) are you using? (129)



48.8%	Self hosted servers
27.9%	AWS
25.6%	Hosting provider
8.5%	Google Cloud
3.1%	Microsoft Azure
3.1%	Digital Ocean
1.6%	Heroku

Other: *CDN (Akamai), Github, Albatross, github pages, private cloud from external party, Not Applicable*

Very few respondents favour Cloud computing services right now (and when they do they AWS seems to be the winner). It is not clear however whether this is due to the love of administering and hosting one's server or because of the lack of integration between the OCaml ecosystem and the service

providers.

2.8 Feelings

Q34: I feel welcome in the OCaml community. (279)

A large majority of respondents feel welcome in the community, an improvement w.r.t to the previous survey (68% agree, 18.2% neutral, 4.0% disagree).

Q35: I am satisfied with OCaml as a language. (279)

While we are sure that some may have complaints about the language, an overwhelming majority is satisfied with it.

Q36: OCaml tooling provides a comfortable workflow for me. (278)

Two thirds of the respondents seem to have found a comfortable workflow to develop in OCaml.

Q37: OCaml tooling is confusing to newcomers. (279)

In the light of the previous answer, this question indicates that the learning curve for OCaml tooling is steep, that respondents (who did not stop using the language) eventually found a comfortable work flow.

Q38: What would you change? (129)

See section A.3 for the raw answers.

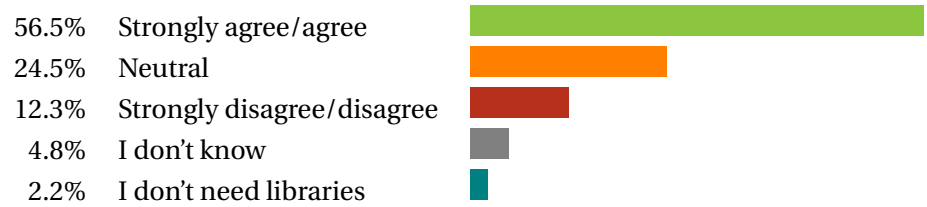
Q39: I am satisfied with OCaml's package repositories. (277)

Again, while there are always things that can be improved, a majority of respondents is satisfied with the package repository.

Q40: What would you change? (57)

See section A.4 for the raw answers.

Q41: I can find OCaml libraries for my needs. (278)



Here the satisfaction rate falls, there are clearly aspects that can be improved w.r.t to libraries for specific application domains.

Q42: If you are not satisfied, which libraries are missing ? (48)

See section A.5 for the raw answers. Many respondents mentioned the lack of GUI libraries as well as Web libraries (e.g. REST libraries, OAuth2, ...), yet in question 2.1, more than 20% of the respondents work on either Web frontend or Web backend software.

Q43: OCaml libraries are well documented. (278)

As in last year's survey, library documentation seem to be a pain point in the ecosystem.

Q44: OCaml libraries are stable enough for my needs. (274)

Opposite to their documentation, the code of libraries in the ecosystem is recognised as very stable by three quarters of the respondents.

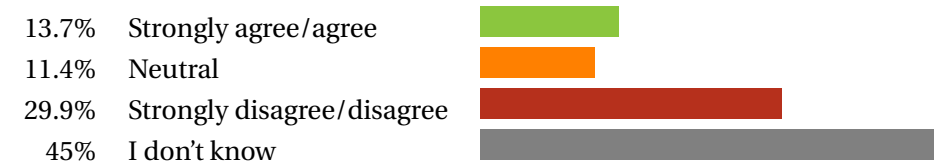
Q45: I have a good understanding of OCaml best practices. (279)

A sizeable percentage of the respondents seem to not understand the best practices, another aspects which can be improved.

Q46: Software written in OCaml is easy to maintain. (276)

A strong point of OCaml, verified year after year.

Q47: As a candidate, I can easily find OCaml jobs. (271)



The biggest percentage of respondents don't know how to answer, presumably because they have not been confronted to OCaml specific job search. For those who have, finding OCaml related jobs seems hard.

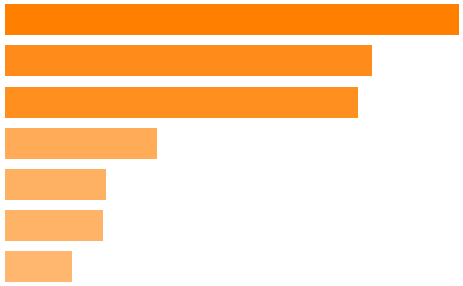
Q48: As a hiring manager, I can easily find qualified OCaml candidates. (246)

An even bigger majority seem to no be in a hiring or managerial position w.r.t to OCaml jobs. But again, for those who are, finding good OCaml candidates seems to be a challenge.

2.9 Burning desires

Q49: If I was granted up to three new language features today, I would ask for (no choice means "no new feature, please!"):

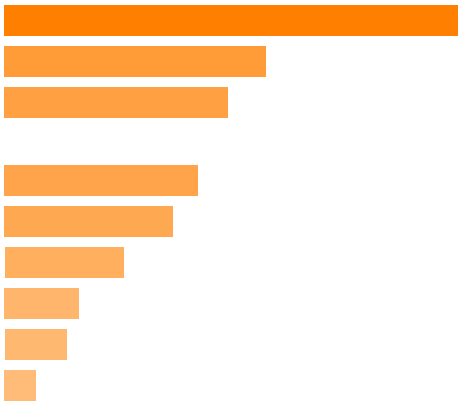
70.4%	An effect system (a type system for side-effects)
56.8%	A builtin "deriving" mechanism
54.7%	Typeclasses/implicits
23.5%	A more powerful module system
15.6%	Namespaces
15.2%	No new feature please!
10.3%	A complete redesign of the surface syntax



The clear winner is the effect system, which is understandable with the release of MultiCore OCaml. The two following most wanted features clearly target boiler-plate OCaml code that could be handled more elegantly (deriving operations from type definitions and writing "overloaded" style functions using an implicit dispatch mechanism).

Q50: If one piece of the ecosystem could magically be made state-of-the-art, I would ask for: (255)

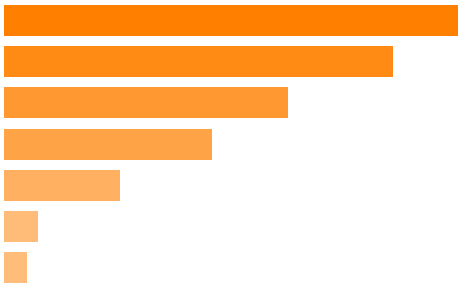
28.6%	Documentation for user libraries
16.5%	A package manager
14.1%	Communication to outsiders (blog posts,conferences...)
12.2%	A compiler
10.6%	A build tool
7.5%	My domain-specific library ecosystem
4.7%	Documentation for the core language
3.9%	A language website
2%	A support community



Here the answers are pretty much spread, with documentation of libraries being again highlighted.

Q51: What do you think are the main pain points that prevent OCaml adoption for new projects? (233)

53.6%	Lack of critical libraries
45.9%	Too hard to find and hire OCaml developper
33.5%	Lack of critical developer tools
24.5%	Too hard to learn
13.7%	OCaml does not fit your environment constraints
3.9%	Lack of performance
2.6%	Lack of backward compatibility



Focusing on the most popular choices: in an industrial setting, the lack of OCaml developper is a clear pain point in starting a new project with OCaml. But more generally (whether in an industrial or in open source projects) respondent seem to indicate that they would not choose OCaml due to the lack of critical libraries. These proportions have not changed much since the last survey (50.4% for hiring OCaml developpers and 51.1% for the lack of libraries).

Q52: Anything to add ? (82)

See section A.6 for the raw answers.

3 Preliminary conclusions

For each of the targeted areas of the survey, we try to list some preliminary conclusions. We highlight issues or improvements that the survey seem to indicate as well as some actions that the OCSF as taken or would be willing to take.

For OCaml as a language

- the language and runtime is seen has mature and solid
- people appreciate the work of the release team and the stability of the various releases
- yet, people also welcome radical changes : the arrival of MultiCore OCaml was a huge demand in the previous survey, now that it is landing[1], people yearn for using it in a safe way through typed effects

With respect to improving the core language, although it is chiefly the work of the core team, the OCSF has tried to finance exploratory work e.g. by funding internship on various topics (such as modular implicits).

The OCaml Community The community is welcoming and one that seems to have avoided, for now, disruptive and toxic behaviours from its members. To ensure that it remains so, the OCSF as endorsed the efforts to create a Code of Conduct, which recently came to fruition with the formation OCaml CoC committee and the publication of Code of Conduct[2]. As for Community related events, the OCSF as sponsored several user centric events (meetups, summer schools, workshops).

For the OCaml distribution One of the recurring pain points is the distribution and use of OCaml (and related tools such as opam) on Windows. The OCSF is currently funding Jonah Beckford's Diskuv project[3]. which has recently reached version 1.0.0.

For the OCaml ecosystem The main pain points that appear from the survey seem to be :

- the lack of various domain specific libraries
- the general quality of user library documentation
- the steep learning curve of various tools for beginners
- the lack of general well documented good practices

It would seem useful to write tutorials for semi advanced users that cover the whole development cycle, from getting OCaml *and related tools* up and running, to good practices while writing code (how to document your library so that it can be exploited by other tools or sites[4], how to distribute it via opam, ...). The OCSF would welcome and fund any initiative by members of the community

[1] And actually has landed since the survey was held.
[2] https://ocaml.org/policies/code-of-conduct
[3] https://github.com/diskuv/dkml-installer-ocaml
[4] such as https://docs.ocaml.pro/about.html

willing to work in that area, has it has done for previous documentation aspects (such as tutorials and improvements to the OCaml Manual by John Whittington).

As for the code itself, the OCSF is funding or has funded several projects that provide valuable libraries (e.g. the Caqti database library, the Dream Web framework). On the topic of integrating OCaml with generic tools, the OCSF has funded the work on demangling OCaml symbols within the `perf` Linux tool.

The survey seems to indicate (particularly for Web related libraries) that a lot of in-house code exists in various projects (from the fact that a sizeable part of respondents deploy Web applications written in OCaml or work on Web frontend/backend as their main project). The OCSF would encourage, in particular through funding, making this code open source and available to the community.

A Verbatim free-form answers

A.1 If you learned in school or University, which one (optional, you may enter the name of your school/University or simply the country). (101)

- Aarhus University
- Boston College
- Cambridge
- Canada
- classe prépa française
- Classe préparatoire du Lycée Masséna, Nice, France.
- Classes préparatoires
- CPGE
- CPGE + Université de Bordeaux
- École Normale Supérieure de Lyon
- École Polytechnique
- École Polytechnique (France)
- école polytechnique, Palaiseau, France
- ENS de Lyon, France
- ENS Lyon
- ENS Paris
- ENSEEIHT
- ENSEEIHT / INP Toulouse
- ENSEEIHT, TOULOUSE, FRANCE
- Epita, in France
- Epitech
- france
- France
- France
- France
- France
- France
- France
- france
- France
- France classe préparatoire
- France, Paris 7
- French CPGE
- French prépa then ENS Paris
- French preparatory classes
- I took an online MOOC from INRIA a couple of years ago. That's the closest I got to learning it at a "school or university"
- INRIA and CEA, LIST
- INSA Rennes

- Lille 1
- Lycée du Parc
- MPSI France
- NUS
- Open University, Israel
- Paris
- Paris 11
- Paris 7
- PARIS DIDEROT
- Paris Diderot
- Paul Sabatier University, Toulouse, France
- polytechnique
- Polytechnique
- Prépa
- Prépa in France
- Purdue
- Radboud Universiteit (Nijmegen, NL)
- Rennes, France
- Rochester Institute of Technology
- Seoul National University (South Korea)
- Sorbonne Université
- Sorbonne Université Sciences (Paris 6)
- sup MPSI et spé MP
- Technical University of Munich (TUM)
- Technische Universität München
- The Johns Hopkins University
- Toulouse III
- TUM
- Umass Lowell
- UMONS, Belgium
- Universidade da Beira Interior
- Universidade Federal de Uberlândia - Uberlândia, Minas Gerais, Brazil.
- Universität Innsbruck
- Université de Paris
- Université de Poitiers
- Université de Rennes
- Université Joseph Fourier
- Université Louis Pasteur, Strasbourg, France
- Université Paris 7
- Université Paris 7 Diderot
- Université Paris Cité (née Université Paris 7 Diderot)

- Université Paris Diderot
- Université Paris Diderot
- Université Paris Sud
- Université Paris Sud
- Université Paris Sud
- Université Paris-Diderot
- Université Paris-Saclay
- Université Paris-Sud, France (now known as Université Paris-Saclay)
- Université Savoie Mont Blanc
- University of Beira Interior (Portugal) and Université Paris-Saclay (France)
- University of Beira Interior, Portugal
- University of Beira Interior, Portugal
- University of Bologna
- University of Cambridge
- University of Cambridge
- University of Cambridge, UK
- University of Colorado Boulder
- University of Orleans, France
- University of Pisa
- University of Wisconsin
- University Paris Sud, France
- UPMC

A.2 What is a pain point when learning the OCaml language?

- 1. The order of evaluation is unspecified. 2. Lots of functions raise exceptions with Failure + a string. 3. Modules in the standard library often do not have coherent APIs and are missing useful features (this is getting better).
- 1. Thinking recursively, 2. Understanding the setup of developer tools (I learned 5 years ago, this has gotten better.), 3. Compiler messages (Still not significantly better. I would like it to be much more helpful like the Rust compiler.)
- 20 years agot there weren't any books
- actually starting to develop real programs instead of simply using ocaml as a language for school exercices
- All blockchain..
- As the first functional language is a hard transition for an inexperienced programmer that is only familiar with imperative languages. Specially given how demanding it's compilation process is.
- async choices
- At least back when I first learnt it, there was very little documentation and example snippets available. Aside for asking the professor for help who was a real OCaml pro, it was next to impossible to debug your code by yourself as a beginner.
- At the time it was packaging and Windows support
- Back in the days, the tooling was very bad.
- Building OCaml projects is very complicated, and there are no good "copy and paste this to get started" examples.
- Canonical libraries for what I need (http servers, event/stream processing, multithreading). I know there are libraries in the community, but would prefer if Ocaml does endorse some projects to start with.
- Choosing between the different features: variant (std vs poly vs GADT) vs object vs module
- Coming from python, something I remember catching me up is that strings are not destructurable/iterable in the same way I expected, so I often needed a function which would explode a string into a char list

- Complexity building own source code
- Confusing type errors productes by the compiler and syntax different from C languages
- Cross-platform compatibility, especially in regards to the package ecosystem and OPAM. Windows is a second class citizen. No cross-compilation. No webassembly support.
- Debugging
- Dependency management
- Developer tooling (much improved since 2017)
- Difficult to debug output. Not obvious which set of libraries to use.
- difficulty inspecting values in debugger and cumbersome printing of custom types for debugging. few tutorials on basic libraries; few and incomplete numerical / scientific library bindings
- Difficulty to set up correctly on my own machine
- Documentation
- Documentation
- Documentation, ecosystem and a lot of packages in opam that are no longer maintained but it's hard to find (when you come from Rust, for example, you're used to import packages and try to do the same in OCaml)
- Documentation/discoverability for advanced/newer language features, such as GADTs
- Ecosystem
- error message brevity & a shortage of soft documentation
- Error messages
- Error messages, but it has been improved a lot (really a lot) compared to 10 years ago
- Error messages; dealing with ';'
- Even the best libraries are not thoroughly documented, the ppx ecosystem is also not well documented. And then there's the opposite end of the spectrum where there's huge academic papers about type theory and no ELI5 explanations for users.
- Everything is slightly broken and incomplete. If it is not broken it is ugly syntax wise. Documentation is always hard to navigate and incomplete. No examples etc. For every basic tool there are X alternatives, all of them with incomplete docs.
- Finding Documentation of the language as well as documentation of how to use popular libraries
- Fragmented and confusing tooling (dune, opam)
- Fragmented learning sources or the real world ocaml using jane street libraries
- Friction between Jane Street and other libraries, especially async vs lwt.
- Functors
- Getting started with tooling
- Getting the development environment set up: which version of opam doesn't operating system provide? How can I install the latest? How do I start a new project? etc…
- Getting used to strong types I guess
- Hard to find examples of idiomatic design and coding
- I find the ecosystem rich in terms of books, docs etc. However, I don't think we have enough courses like other communities.
- I think bugs due to grammar issues such as 'then…;'
- I think we need a central point for the documentation of all OPAM packages, in one place and searchable
- If you are not familiar with functional programming it may not be easy.
- installation on windows
- Integration at library link time is way more messy than it should be.
- it needs better documentation
- It used to be 'tooling/IDE setup' but I think the only pain point left is that FP takes a while to learn (so it's not an OCaml Issue, but it does hurt)

- It's hard to use on Windows, and errors are not always explicit.
- Knowing which abstraction suits better to my current problem (raw polymorphism vs object polymorphism vs modules).
- Lack of a built-in "deriving" mechanism.
- Lack of a cohesive guide to getting started, how to organise projects, how OCaml projects are typically structured, do I use functional or mutable features at an architectural level.
- lack of a cookbook to help beginners know how to do simple things
- Lack of ad hoc polymorphism (using print_string Vs print_int felt like a bit of a bad joke)
- Lack of books / resources / tutorial
- Lack of books and examples for advanced topics (GADTs, phantom types, ...).
- Lack of centralized documentation, official manual can be cryptic.
- Lack of debugging tools, even the dumbest one: polymorphic best effort printf
- Lack of documenation, video lectures on best practices
- Lack of documentation and beginner friendly articles
- lack of documentation for the most arcane features (that I suspect I do not need anyway)
- Lack of documentation, unfamiliar syntax, fragmented and inconsistent tooling
- Lack of good tutorials
- Lack of high quality tutorials
- lack of industry adoption, I would learn it much more if I could do it as my job
- Lack of libraries, tooling not on par with popular languages
- Lack of online courses on popular training platforms (coursera, etc)
- Lack of other developers
- lack of out-of-the-box support from IDE (but admittedly, this was a long time ago)
- Lack of peers willing to use OCaml
- Lack of real-world tutorials using the common developer tools
- Lack of resources
- Lack of tutorials
- lack of tutorials
- Lack of tutorials for advanced features
- Lack of visible syntactic scaffolding and all the implicit stuff that the compiler is clever to figure out. But, it's also what makes it great once you do learn it.
- Lack of widespread community support
- lacks of tutorial for basic programming skills and takes long time while doing opam upgrade
- Learned a long time ago. Today I think that (compared to other programming languages) a major pain point is the compiler errors being arcane (not including hints to resolve, not linking to relevant part of the documentation, being very academic, etc.)
- Learning how to build projects with multiple files
- Learning how to use build tools
- Learning it is not easy like learning front-end skills
- Libraries with auto-generated docs that only show the API interface and don't give any usage examples.
- Library and doc got much better but tooling is very limited.
- Like of good resources for advanced features (GADTs...)
- Maps, hashtables, etc.
- Maybe add something like Rust Traits to enable easier print-style debugging?

- module and functor syntax
- Modules provide too little documentation
- Monads
- Monads, but once you get them it's so nice!
- Navigating the available libraries (both identifying them and accessing documentation)
- No
- No clear preference for specific libraries (e.g. a canonical standard library)
- No IDE.
- No one
- no plug-and-play editor/compiler combo makes for annoying "unknown module Foo" at the start of projects when you don't understand dune subteties
- Non cohesive tool chain for project a management
- Non OCaml developers
- None
- None
- Not any suitable examples over the internet. I feel like I always have to reinvent the wheel!
- Not sure. I've been coding in Python for over 25 years. I'm having trouble learning OCaml.
- Nothing particular other than that the community is small.
- Originally it was tooling and documentation, but those are getting better
- Packages availables
- Pattern-matching, Functional paradigm
- Platform, toolchain
- Poor syntax errors
- port packages for crossplatform compiling
- ppx
- Print-debugging is verbose and clumsy (and especially without relying on ppx)
- quirks in the syntax : - everything should be a 'let ... in' with a 'public' keyword for exported values - 'end' should be mandatory for 'if' and 'match'
- Reduced amount of up-to-date books
- Resetting your brain to the functional paradigm.
- Scattered documentation, too many too complex tools badly documented (they are extremely flexible which is good, but the documentation is really terrible, also dune's)
- searching for doc on cryptic ocaml syntax (such as 'type = ..') when you don't know what it refers to.
- Setting up a dev toolchain is slow and requires knowledge; also really hard on Windows.
- Setting up a working OCaml enviroment, even on Linux is incredibly painful. Other languages like Golang, Python, you can install from the repository and be ready to run. OCaml is constantly getting errors, and makes it difficult to learn.
- setting up new project with opam files, dune, working switch with language server
- Setting up the build system and accessing standard libraries
- Signature vs usage syntax discrepancies (: vs = in records and named arguments)
- Sometimes hard error message to decode and a lack of online tutorial videos.
- Strange types: lack of uint, 31/63 default int types. Poor standard library.
- Stratificiation of language features, a cleanup would help
- Support on Windows, tools that are not as good as in other languages (e.g. debuggers)

- Syntax (only at the beginning)
- Syntax and tooling schisms Not enough well funded libraries/frameworks Immature web and mobile development story
- Syntax error messages
- Syntax is so different from popular languages
- syntax of match expression
- The community and general toxicity towards questions.
- The documentation and tooling
- The documentation is missing examples. A lot of the API doc would benefit some examples that could display basic usage of the functions.
- The ecosystem
- The expression based syntax, when you come from Python for example. You can't just willy nilly create variables or if/else statements that don't fold back into a value.
- The fragmented concurrency landscape.
- The interaction between Dune and OPAM and sometime, lack of documentation
- The lack of "end-user", application-oriented tutorial
- The learning curve of dune, for properly structure one's first ocaml project using this build management system.
- The let in is scary. Too many manuals/books referencing to different standard libraries. It's not a big deal when discussing assyncronous effect types sincr I'm used to different implementations in scala, but having that confusion at the base layer weird.
- The match and if syntax
- the new way of thinking. better at last
- The Real World OCaml book uses Jane Street libraries, which not all projects use, so there's a surprising transition for beginners when they go from learning to actual projects.
- The relationship between OPAM, Esy, Dune, and the OCaml compiler is quite confusing at first. Contrast this with cargo/rustc.
- The resource available is extremely limited, especially sample projects that are easily accessible like web or desktop applications. I mostly find discussions on language design or type system or formal method, not something resembles a day-job projects.
- The sometimes cryptic type errors
- The standard library is small, therfore how do you do stuff "the right way"
- the switch between REPL work and multi-file projects
- the tooling
- The use of the 'in' keyword
- There is no official Korean book in my country.
- There seems to be no easy to find single source of truth for documentation. I'm glad the community is working on fixing this issue
- Things that might be simple in imperative languages is a bigger process in OCaml
- Thinking in the riht way for functional programming
- This is almost 20 years ago, so take it with a grain of salt, but the lack of documentation of advanced features makes things difficult somehow - I always have the feeling that I'm only using a small part of the power of the language.
- toolchain
- tooling and fragmentation
- Tooling and setup (disconnect between opam, dune)
- Tooling and tooling documentation. Stdlib
- Tooling is very raw
- Tooling, instructions for setting up and building a first, basic program
- Tooling, lack of a proper standard library, lack of concurrency.

- Tutorials of the building system.
- types
- Uncertainty about the syntax: toplevel declarations, unclear priority between if and ;
- Understanding type errors and their error messages.
- Unicode translation OCaml <-> Python; it turned out to be a non-issue.
- Unit testing.
- Used to be the many options for build systems
- Vague compiler messages
- Wanting more complete ocamlverse to steer which libraries that I should learn and use
- Weak documentation, awkward syntax
- When I first learned it in 2017, I tryed to install OCaml on windows and it was a nightmare to get both OCaml and an editor working. I did not manage to do it, and kept doing OCaml in school only until I got a linux machine.
- Windows installation
- Windows support
- Windows support
- Windows tooling
- Working with modules and using them effectively in building regular projects eg web backends / cli applications
- zero parameter functions, I/O channels, usage of semicolon

A.3 OCaml tooling is confusing to newcomers/what would you change ? (129)

- 'end' for match expressions
- 1. Compiler messages. Errors need to be much more explanatory/suggestive of problem/solution like rustc 2. Fuller, friendlier tutorial getting up and running on developer tools (as a set), understanding the developer ecosystem for beginners.
- A bit more emphasis on "official" ways to do things would clear up things for beginners in my opinion. Especially for project structure.
- A built-in "deriving" mechanism.
- A good deal of the stumbling blocks I encountered when starting in OCaml have been removed, and I think the ecosystem has been trending toward approaches that newcomers will find easier already. Keep it up!
- A unified tool for package management and build-tools will simplify the experience for new users. Having relocation support in the compiler will also be great as that will open the door to re-using the same compiler for many different switches.
- Add clear MWE in tools' docs, as well as common errors and how to avoid them
- Allow more than one version of a package in an opam switch to allow for locked dependencies.
- At the very least, built in support for deriving pretty printers. Ideally compiler support for deriving a run-time type representation and the ability to write additional type-driven plugins.
- being more opinionated with tooling. look at golang. they have a single command „go“ which lets you build code, handle dependencies, tests and benchmarks and much more.
- Better debugging tools
- Better debugging tools
- Better dev workflow on Windows
- Better documentation and examples
- better documentation, especially for dune and opam, and how to publish packages, documentation about best practices
- Better dune documentation
- Better examples of workflows.
- Better integrated 3rd party packages like Go

- Better integration between dune and opam to provide a single entry point into installing and building OCaml projects. Something like rustup/cargo which covers building and installing libraries, and updating rust versions in one binary cli.
- Better integration of the tools (and reduce the many layers under the hood). Better documentation generation.
- Better macros support - using the ppx AST interface is a byzantine maze and requires frequent checks to the API, making the macro code hard to understand and maintain, wheras in languages like lisp, macros can be written as easily as normal functions.
- Better support for higher-kinded polymorphism; higher-rank types; row polymorphism.
- Better support for windows
- Better Windows support, we cross compile from Linux to Windows which is cumbersome installing third party libraries
- Communicate about recommended tools and good practices ; make tools 100
- Creare something like rust doc or goexamples
- Create a project scaffolding tutorial. Dune and maybe some other tools that are good practice for the majority nowadays
- Create a windows version easy to install
- Debugging
- Declutter the language (remove objects, labels, type system cruft, etc)
- documentation could be more extensive
- Dune and Opam using JSON
- Dune, opam or esy... Just pick one! Stack is wonderful pour Haskell application, and it does almost all the same things...
- Dune. Most of it.
- Dune's weird love-hate relationship with global state. Too tailored to quite specific kinds of users.
- easier multitier programming tooling. Merlin, dune, etc... is restricted to single tier and hard to make work with more than one compilation output/lexical environment
- Easy cross-platform build (like Go, Rust). Better cross-platform ecosystem.
- Endorse best tooling practices for each audience (web vs standalone binaries...)
- First class support for Windows
- Fix the vscode plugin. I often need to manually restart the language server.
- focus all efforts on dune with a cargo-like package management workflow (lockfiles, immutable repo, etc.). opam could be a library used in dune for constraint solving capabilities and that's it.
- Focus on standard library to strength the eco system
- Get a more unified tooling story. Remove the distinction between libraries and packages. Nowadays packages only install one library. Write library version constraints in dune files. Stretch idea: instead of version numbers, depend on interfaces.
- Good tooling to understand performance bottlenecks
- great "official" getting started guides that are endorsed. If that's not possible, multiple guides that clearly show up-downsides so that users can make an informed choice
- Have dune or ocamlbuild a part of the standard distribution so that I can use it when shipping software to end users that are not familiar with OCaml.
- Have more collaborative communication between the developers of core tools
- having one 'blessed' toplevel wrapper around opam + dune + ... like drom or other
- Having opam and dune combined into one tool. More importantly though: Being able to install native dependencies (zlib, openssl, ...) with opam. This is currently only possible with esy (which is mainly why i am using it).
- Howto document on www.ocaml.org explaining history and current best practices. + FAQ: What is ocamlbuild? jbuilder? dune? ....
- I don't know
- I don´t know what to change. But right now I'm trying to learn about camlp4,5, ppxlib, ppx rewriter. Macro/meta-languages are hard enough without this confusion :)
- I need *one* stable tool to build/test/deploy/configure/reproduce/bench in an *easy* way, even for casual programmers (I program only a couple of months every year so the current tooling is powerful but too hard to use, even more so for students).

- I still find it hard to understand the relationship between dune files, modules, and layout of code files on disk. I think it is partly because I come from scripting languages where there is typically 1:1 relation between files layout and program modules.
- I think that more comprehensive guides will help newcommers
- I want a STATELESS packaging and building approach, reproducible. I don't care how it works, and I also don't care about whether packaging is separate from or part of the build tool. I do not want a tool that has commands that change the state of system.
- I wish there was a simple setup for VS Code that didn't break. I got it all working a few months ago and took a break from OCaml, and now it's all broken again. Ugh
- I'd like to finally have a tool that's endorsed by the foundation where you can "tool init project" "tool build project" and "tool exec project" easily like cargo for Rust
- I'm not sure, but I think the S expression configuration format dune uses is something that takes getting used to. Also some new users seem to be confused about what opam does and what dune does.
- Implicits to make all those dots on arithmetical operators go away. I appreciate why people like typed operators, and it should be an option–I like it in some contexts–but it makes mathematical code ugly.
- improved documentation of libraries and tooling
- Initial tooling setup remains too complex for beginners; simplify/unify.
- Installation packages for Windows, MacOS
- Integrate the functions of dune and opam into a single tool
- It should be easy to define syntactically new operators
- Language: new features (it's already coming along well). Tooling/Newcomers: consolidate a standard set of tools and write comprehensive guides for them.
- Language: support for unicode in identifiers; ability to use more infix or misfix notations; ability to canonically generate combinators associated to types (but I did not check the most recent versions, so this might have changed)
- Languages like Rust have raised the bar with regards to what newcomers expect from tooling, particularly when it comes to package management and build systems. OPAM was great some years ago, but it's starting to show its age.
- Less dune dependence.
- Library names are hard to remember: 'foo_bar', 'foo-bar', 'foo.bar'
- Make an IDE that replaces all of the command line options and s-expr configuration files with a GUI and has a built-in top level.
- Make dune do the opam related stuff as well so beginner can only learn one tool.
- Make it easy to setup and use. The first time I started to learn OCaml i setup ocaml on my home server, got nothing but errors and errors so I gave up. When I came back to Ocaml I used a web compiler to learn it. I haven't attempted setup again, yet.
- Make it more beginner friendly
- Make it significantly less painful to set up correctly for most use cases. Even minor pain points can drive a newcomer away for a long time or permanently.
- Make one opinionated tool that does all (installs, builds, published), like cargo
- Maybe add support for F# active patterns and type providers
- Merge Dune and OPAM, significantly simplify Dune documentation
- merge package management (opam, findlib) and build (dune). zoo of tools is a barrier to newbies.
- More and better documentation
- More beginner friendly and production ready frameworks that scale with your app (for example Dream)
- more comprehensive "getting started" walkthroughs with different setup options (e.g. repls).
- More documentation and tutorials for tools, especially for Dune.
- More self-contained guides!
- More tutorials would certainly help (I should contribute more on this front), and I think the RFC 17 could also help make the story clearer to new comers.
- More welcoming tutorials, articles, documentations...
- Move the functionality of OCamlFind into the compiler project.

- My main concern is debugging: ocamldebug is unreliable; gdb and the like are unusable. As for memory consumption issues, I know how to gather some data, I don't know how to easily exploit those data.
- Nothing !
- Nothing. What's coming in the future is really exciting, everything we can wish for are being planned.
- One of the reasons I haven't gotten into things like dune more is that I don't feel I will be able to easily put together a simple project to get myself started. A good tutorial might be helpful.
- One tool to unify the workflow
- One tooling front end, like cargo
- Opam and dune need more examples and thinner manuals
- opam: (1) use Nix style instead of global switches and (2) support network file system. dune: better documentation
- Provide a step by step Lwt-aware debugger. Make dune support test that depends on other tests (tests can only depend on production libraries at the moment).
- Provide an unified front. Improve opam and dune stability and portability.
- Provide some new-comer friendly info
- Remove overlaps between opam/dune/ocamlfind somehow.
- Requiring opam to start learning the language.
- Revive ocamlcc
- Robust single-binary CLI front-end for build system, package management, project initialization etc.
- Simplified setup and use of opam, dune, ocamlformat and lsp
- Some basic ppx should be merged into the compiler.
- Stop making the language less accessible to beginners (ex. ppx)
- Stop teaching the tutorials by starting on the repl since it's confusing and explaining the difference between botmal code and the usage of ;; is hard to understand. People don't program on the repl. Start things with a proper project and tooling.
- Support circular linking relationships between files & forward references within files without using "and"
- The current situation around libraries and packages is confusing, follow dbuenzli's RFC to solve this.
- The dune documentation at least
- The opam and dune docs need a re-write from someone who is good at technical writing I think. I feel they're not easily searchable and give unwarranted emphasis to uncommon workflows and are not really kept up-to-date either.
- the tight coupling between the dependencies of an ocaml project and dune's version is confusing (e.g. we don't have similar issues in Java projects with Apache Maven)
- There could be some kind of integrated IDE with all the tooling bundled.
- This whole industry-wise practice of developing language specific tools and language specific "communities".
- tooling and installation
- Tooling,tooling integration, documentation(etc. Switch creation with latest opam + variants)
- Unified tooling
- Unify the various tools (dune+opam) to provide s single interface
- Unify tooling. Complete docs with examples. Don't have X alternatives running in parallel. Deprecate X-1 of all alternatives. Make a ocaml distribution with ONE set of tooling. Emacs support is always slightly broken and incomplete.
- Up to date getting started, comparison of main libraries with examples. I think ocamlverse is doing the right thing here
- VSCode OCaml Platform plugin regularly breaks when updated
- We need a proper debugger ASAP. It would also be nice to have a blessed wrapper for dune/opam.
- We need a step by step debugger that actually works. Something with a GUI (really, it's 2022!) and preferably (a limited amount of) time-traveling debugging (just a "debug_trace" statement that saves call stack and variables would be a good start).
- Web is king. Make wasm, webdev easier.

- Well you have to learn a whole ecosystem, i feel like its a big overhead to getting anything done. I am not proficient with OCaml tools.
- Who did think that LISP was a good syntax for dune??? Also, dune lacks some modularity for custom applications.
- Windows support; creating opam switches is too slow.
- With unlimited manpower I guess? Make sure that one can find a single debugging and profiling tool that actually does the job.
- Would like to configure dune builds in OCaml itself.
- Write code examples in the documentation

A.4 I am satisfied with OCaml's package repositories/what would you change ? (57)

- A manual approval on a PR for opam is always a bit weird, and forbid very fast fixes.
- Add cryptographic signing, like proposed in the Conex project by Hannes Mehnert
- Adding a new package or updating an existing package in opam is very slow, requires a manual step.
- As above (better Windows support)
- better documentation
- better libraries
- Better packaging of non-ocaml dependencies (like C libraries)
- Binary packages, Avoid inplace modification of packages in opam repository
- Built in cross compiling
- Cargo docs style documentation hub.
- Documentation for many packages is either incomplete, hard to find, or missing
- Easier access to package documentation
- Easier to contribute releases, immutable packages
- Examples!!
- First class support for Windows
- flag packages as Windows-compatible or not
- For a newcomer it is not obvious how to publish a new library / package to opams repository.
- github coupling within standard tooling
- I tend to think that it would be good if we could have something to generate OS packages that do not require opam *for end users*. End users do not want to type commands to install their software.
- I wish more core libraries would be available with the latest compiler version right away. The reason is because I'm interested in the MultiCore capabilities
- I wish opam were more inspired by nix/cabal in that multiple versions of a package can coexist, thus eliminating the possibility of failed upgrades and version conflicts
- Immutable repository
- Improved support for binary packages.
- It is not always evident how to use opam with under-development software (e.g., in a git repo). Better documentation for such cases
- It just work. Thanks.
- it's understaffed and needs more maintainers
- keep the cross-repositories in sync with the mainline, or improve cross-compilation capabilities to remove the need for cross-repositories
- LIbrary completeness and documentation, particularly examples, lacking
- make it immutable, make all the lints from its CI also in 'opam lint'
- Make opam much faster

- Maybe having an indicator about how a repo is maintained. Currently, I'm looking at the last commit on the GitHub repository.
- More bindings to libraries, especially GUI. But also having a nice Erlang/distributed systems library would be nice.
- Most packages are broken due to bad version bounds. Even publishing my own package is a pain because I always need to fix dependant packages.
- My projects (binaries) often break due to changes in dependencies.I need a simple way to prevent this.
- New releases of packages on Github don't seem to be in Opam as soon after release as packages in other languages are in their repositories. I'm not sure what the problem is. Do new uploads to Opam all have to be approved? If so maybe this needs to change?
- New v3.ocaml.org with integrated docs is something I've wanted for a long time. Better UI and integration between dune and opam. More standardisation on using dune for builds.
- not much
- Nothing, it seems expensive already.
- Nothing, the work on opam-repository is truly great!
- Nothing! The OPAM repository is one of the best efforts I ever seen, maybe comparable to Debian repositories, or maybe even Red Hat.
- Opam consistently breaks on Linux, it needs to be stable across new versions.
- Opam dependency hell (better backwards-compatibility culture).
- opam is just so damn slow
- Opam repo as immutable
- OPAM sometimes becomes very very slow for me.
- Opam still unsafe to use on Windows, even with wsl.
- opam's workflow for enabling options like flambda, musl, etc. is worse since OCaml 4.12. Before it was "opam switch create x.y.z+musl+static+flambda" and clearly listed by opam, now it's a less clearly documented set of –package=foo,bar,baz flags
- Organize Opam names in a hierarchical way like CPAN (XML::Light, XML::Twig vs xmlm, pxp, xml-light). Make documenting (writing and publishing) libraries easier (as easy as POD), with clear documentation organization guidelines (as in CPAN :).
- Publishing to Opam is cumbersome, publishing should be faster without manual PRs, the CI could be a later step
- The ability for reproducible opam switches.
- The repositories themselves are great. It'd be wonderful to have more, and more well-maintained, things in them!
- The web programming stuff is a bit of a mess to be honest, but I don't know enough to suggest how to fix it.
- There's a lot of useful libraries and tools packaged in OPAM, but it can be hard to find which ones I need. There should be a (manually curated) short list of "quality" libraries, where "quality" = often useful + well documented.
- Too many packages no longer maintained, hard to find what you want with the small descriptions, there are no tags for packages so you can't filter them like this. It's just a maze/mess
- Very hard to find anything. Bad docs are standard for almost all ocaml packages.
- Windows support, of course ;)
- Would be less strict about possibly breaking people's pin sets

A.5 If you are not satisfied, which libraries are missing ? (48)

- A good web dev story - React + JSOO or something. You know... "frameworks" type of stuff.
- A more integrated Web stack
- a simple & low-dependency async/io/networking layer.
- A well maintained cross-platform GUI library for desktop applications. Revery and Rust's Sixtyfps are good examples.
- Actually I hardly use any libraries except for the standard library. There are changes to be made, of course, but I can always write what is missing myself.
- API bindings.

- async lib
- Aws clients
- Better builtin string functions (even Str's API is somewhat lacking) ; maybe a Shell-like library to replace some usages of Bash/Python ; but the language will never be ideal for those tasks anyway.
- Bucklescript bindings to popular JS libraries
- Cloud host APIs like GCP and AWS interfaces
- Common things tend to be there, but the smaller ecosystem means niche things don't exist. Options for things like GUI and TUI libraries are also pretty lacking, or the ones that exist have some pretty rough documentation.
- Do not, if they are missing but I tend to reduce dependencies and hence I do not always search for libraries and prefer to develop mine
- Even though there are libraries, they look to me unpolished, under-documented, and a lot of them are single-person projects.
- excel writer
- Flutter-like framework
- GUI
- GUI
- GUI libraries are not there yet. There's abandoned bindings for everything but nothing that seems to be used by anyone.
- I can easily find some libraries, but they're almost personal libraries. I want some rigorous, standardized, stable ones backed by some organizations.
- I can't figure about what is missing.
- I dont usually have a lot of dependencies anyway
- I tried to build a stack of web libraries but one required Lwt and another Async.
- In almost every niche we're missing libraries. Personally I would love to have a high performance data streaming library for treating channels and string streams alike.
- libbacktrace equivalent.
- Libraries are often missing, but I can just write my own libraries.
- Libraries aren't missing, the overall quality of libraries is what's lacking
- Libraries to make native multiplatform GUIs
- matplotlib, GR framework (scientific visualization and plotting), complete bindings to recent GSL, julia interoperability, Stan, MCMC libraries such as PolyChord
- mature grpc client library
- Missing or incomplete or immature libraries for RPC like GRPC/protobufs/thrift/avro. Building web pages with JSOO, options like Bonsai are poorly supported while Ocsigen is quite invasive. Native bindings for AWS/Azure/GCloud.
- ML
- more web app building libraries (like what is available in the Rails community)
- More web related libraries
- Most libraries are old, outdated, incomplete and has bad documentation.
- nntp, discord
- non-web easy to use GUI tooling
- Often, the library does exist but it's not maintained anymore.
- ORMs, drivers for databases
- Performant scientific libraries.
- Problem more to do with incompleteness of libraries
- Qt or other desktop toolboxes
- Sdk library's for common sass services and applications specifically databases

- SQL DSL
- Systems programming, graphics, general gaming related
- There's just a lot of random stuff that I reach for and it isn't there. Automatic REST API stuff is probably the most prominent recent example – ocaml-swagger exists to do what I was looking for, but is unmaintained and only supports a now-deprecated API specification language. I had to implement a bunch of OAuth2 flow stuff recently. I've written a few font format parsers. It's just kind of a sparse ecosystem.
- TUI library like Rust's tui. Lambda-term requires 30+ mb of camomile resources redistributed with the binary. Nottui didn't work on windows for me and has too functional api.
- Up-to-date bindings for gobject-introspection libraries and bindings for openapi.

A.6 Anything to add? (82)

- - streamline docs: search by types (hoogle), jump to implementation from docs (hackage), tools that encourage soft-documentation - release tooling: first class debugger, profiler, race detector, cross compiler, executable optimizer, tarball release tools
- "The main pain points that prevent OCaml adoption for new projects": for many people, not having a Windows-first approach might be an issue. Not for me, but compared to "all-in-one" IDEs such as Eclipse, Visual Basic, etc, it's harder to get into.
- > What do you think are the main pain points that prevent OCaml adoption for new projects? [x] Missing marketing, hype, and making devs believe they're the cool kids by using ocaml.
- 1. Documentation for functions in std/user libraries should have examples, types are not enough (Basic tutorials for user libraries should be encouraged also.), 2. More versions of functions returning options/results rather than throwing exceptions.
- A big thanks to the core team and all people working on the tooling (dune, merlin, odoc, etc)!
- As I mostly write code in Python at work, I realized that printing values in REPL is essential to the newcomers. I hope the modula implicit would be adopted as soon as possible so that it makes OCaml print in a more easy manner.
- Better debugging experience, so instead of print_endline we could have breakpoints and see the values (at least in bytecode)
- Better macro support please
- Despite two decades of experience with Haskell, Common Lisp, and other niche languages, I found the OCaml learning curve so steep I gave up twice. Learning the language was easy, learning how to use the tooling to create and build a project was very hard.
- Easy cross-compilation would make embedded development much easier. Compiling via ocamlcc used to work, but last I looked, it only supports <version 4.0. I'd be willing to upgrade it if I could find documentation on the bytecode changes since then.
- Ecosystem is poor. Either a library doesn't exist or is not supported anymore
- Get 'em young. OCaml should be taught to teens in school, at least somewhere
- Have a nice day
- I am looking forward to OCaml 5.0 with multicore support.
- i do love OCaml but I am finding it hard to go from hobbyist to professional (in ocaml) at this later stage of my career ie i cannot really take a big paycut and jobs are easier to find for the standard 'enterprise' languages
- I hope I can bring Ocaml to my work sooner than later. The multicore progress might make it possible....
- I like the effort being put the new docs pages for libraries and the redesign for the ocaml page. Create a central point of contact and learning and stop spreading efforts and knowledge around. It has been hard to find correct, up to date docs to learn.
- I love OCaml
- I think there's nothing wrong with OCaml itself, it just suffers from small community... and perhaps a bit from being small but not also new? The multicore work looks super promising and I hope that can get a bit of buzz going via benchmarks etc.
- I wish I could have picked both "build tool" and "package manager" in what I would like to be state of the art. Designers think of these as separate, but to the user, they are not separate.
- I would love to see OCaml for microcontrollers working alongside C++. Not like OMicroB which requires you to make the application entirely in OCaml.
- I'm only 3 months into learning OCaml through my outreachy internship, so I left blank the questions I felt I was not experienced enough to answer!
- If one piece of the ecosystem could magically be made state-of-the-art, I would ask for: a debugger
- In 25 year of OCaml programming, I considered several times to change for a more mainstream language but I never found another language that would make me more productive. The key ingredient is strong static typing, and OCaml being multi-paradigm.

- It would be good if the next survey were also asked more about OCaml in education. There are many questions about developing (for web, developer tools, repo, hiring, etc). But I use OCaml in education and use OCaml to explain some particular concept.
- It's a niche language with literally no jobs in my region.
- js_of_ocaml should be made the official js compiler
- keep up the great work!
- Last question (main pain points that prevend OCaml adoption) It's silly, and not something I would fix, but people seem to really dislike the syntax at first. I think 90% of people are put off by it upon initial contact with the language.
- Long life to OCaml!
- Missing libbacktrace; see `http://refpersys.org/`
- most of the core tools (dune, opam, ppxlib, ...) are understaffed. They need more people working on them.
- My burning desires are generic printing, an integrated top level (REPL) and JIT compilation.
- Need a symbolic visual debugger that can accurately display values of any type.
- OCaml community is one of the friendliest and welcoming that I've encountered
- Ocaml developers go their own way so it looks crazy from the outside. I'm fine with that as I like to use Ocaml and am fine doing things my own way.
- Ocaml grows with highlighting type safety and webdev.
- OCaml is a great language, thank you for supporting it!
- OCaml is a nice language, Though in my setting I use it only for prototyping.
- Ocaml Is Amazing
- OCaml is amazing and you did a great job building a nice community with highly focused members which are incredible dedicated in moving the language/eco system forward. Please streamline the tooling like golang did and provide a language server
- OCaml is growing everyday !
- OCaml is looking good, but there's a long way to go. Without typed effects, we're making the language a lot less secure. Proper debugging is also critical ASAP: no language is debugged via printfs anymore.
- OCaml lacks a stable, self-contained, easy to use (even for non daily programmers) tools and productivity-enhancing features (a "deriving" mechanism as PPXes are in my view really problematic (incomplete, buggy, barely documented...).
- Ocaml needs more programmers than anything. Even terrible languages make irreplaceable progress through large amounts of public attention.
- ocamldebug for native code would be nice. Toplevel for native code also.
- odoc made a lot of progress but it still needs to be made more reliable; I miss a modern literate programming environment (useful for writing examples in doc without copying code in .mld files)
- often clients put constraints to technology
- Overall I'm very positive about how OCaml is developing and the direction of the community. The pending release of multi-core and future typed effects will be hugely important. Plus improving opam/dune/merlin will help attract new developers and existing.
- Please add unboxing (and block pinning for C interop) so I can entirely move on from F#!
- Point 1: ie Windows
- Reading documentation for functorized code is very hard
- Some things that prevent me from advocating for adoption of OCaml where I work: no good tooling support for Bazel, poor coverage analysis tooling, the PPX ecosystem is way too fragile.
- Thank you for organizing a survey!
- Thank you for the survey !
- Thank you for tracking this feedback!
- Thank you for your work to improve OCaml!
- Thanks
- Thanks !

- Thanks for a great 2021!
- Thanks for all your work supporting the ecosystem!
- Thanks for running this survey!
- Thanks for your work!
- The community is toxic, a recent example URLTOSOMEDISCUSSPOST This is a response I would expect from new members and not maintainers, its why I avoid posting.
- The debugging experience leaves much to be desired. This is not specific to OCaml - debugging native languages in general seems difficult. But my point of reference is Chrome Dev Tools for JS, which I think is state-of-the-art for debugging.
- The idea to have a list of "blessed" libraries for newcomers is good. It should involve the community, for instance with a voting system.
- The lack of diversity and representation is striking.
- The main issue with adoption is marketing, not the language or tooling!
- The OCaml experience has drastically improved over the past few years. For instance via VS-code/ocaml-lsp-server, ocamlformat, dune, opam and improved docs generations. Excited for what to come next!
- The onboarding experience in OCaml is pretty poor (unclear, unfamiliar, confusing tools), improving that would make a big difference
- The one thing I'd really wish for is a state of the art garbage collector. It must rival or better .NET and Java, and must never stop the world
- The pain with OCaml is usually that there are way too many breaking changes between language versions - some sort of mitigation for that would be really helpful.
- The piece of the ecosystem I desire the most is a step by step Lwt-aware debugger, like you find in any mainstream OO language.
- There is some kind of low quality mindset in the OCaml community that must be dealt with.
- Too hard to learn because of environment setup. Even setting up VScode takes a lot of time and is fraught with errors. I want to spend more time doing with the language, and learning its concepts / workflow not fighting errors for the compiler to work.
- tooling and lack of standard deriving are two main reasons why people might pick rust instead of OCaml even though OCaml is higher-level. Syntax is also a bit weird. Reviving reason (!) as a syntax that is easier to format, fewer warts would be good.
- Tools written in OCaml are often too hard to install by non-OCaml developers if they cannot use their favorite package manager (assuming it even exists). In particular, Opam is too OCaml-specific in that respect
- Typeclasses and deriving mechanism please!
- We badly need a good developer onboarding experience on ocaml.org, including project setup with opam and dune. Right now it's very difficult for newcomers to figure out the current recommended practices for how to put together OCaml projects.
- We need a GUI debugger that just works. Refactoring tools (e.g. renaming incl. references in other modules). Easier definition of printers, serialisers, comparers (get rid of the broken = and Compare). Better row polymorphism (for nanopass compilers).
- We suffered from the Lwt/Async schism, and I reckon that OCaml multicore + effects will be a unique opportunity to bring unity. I would love to see some of the community heavy-weights resolving shortcomings with proposals such as Eio before it's too late.
- WINDOWS SUPPORT : Clear instructions (one place, one source) for compiler + opam packages (automated + manual for people behind firewalls).