**Directions of next generations programming languages**

**Intro**

* The goal of this paper is the conception and implementation of a new programming language called Rym
* What exactly is a programming language
* artificial language
* turing complete
* general

**General trends**

* more powerful type systems
* type inference
* traits/type classes
* easily extend functionality of uncontrollable 3rd parties
* JavaScript
* TypeScript, other one from
* Facebook
* Proposal to add type annotations
* Python
* added type annotations
* Rust, Go, Swift
* auto in C++
* state of C, C++, C#, Java, php types?
* jit compilation?
* null safety
* Rust, Swift, Dart
* TypeScript checks for null/undefined
* what are the others up to?

**Factors improving adoption speed**

* (save) ffi

**Language Goals**

* simple and clear design
* as simple as possible but still enough features for great dx
* no magic
* complex features build on top of
* simpler ones (composition)
* user is able to look at and understand everything from high to low-level implementation details
* no macros?
* names are easily understandable
* const: constant
* mut: mutable
* abbreviate only where it makes sense (items that are used veeery often)
* not too specialised
* allow multiple paradigms
* turing complete
* build on top of existing work
* see specific new features

**Specific new features**

* algebraic data types
* enum variants can contain data
* simpler borrow checker
* const/mut variables
* ex-/internally mutable function parameters
* external: argument must be passed mutably
* fn test(param: mut Type);
* call(mut arg)
* internal: passed in value will be copied and can then be mutated
* fn test(mut param: Type);
* call(arg)
* postfix unwrap operator (!)
* condition: lhs type must implement Tryable
* calls Tryable::unwrap(self)
* success variant: returns data inside
* fail variant: panics with an appropriate error message (should contain data from fail variant if there is any)
* Examples: Result::Err, Option::None
* postfix try operator (?)
* condition: lhs and function return type must be the same
* condition: type must implement Tryable
* if lhs evaluates to a failed result evaluation, the enclosing function immediately returns the failed value
* optional chaining (&.)
* condition:
* lhs and rhs produce the same type
* lhs can be downgraded/coerced to rhs (Result into Option)
* when lhs evaluates to a failed result, evaluation of chained expressions is stopped and the entire expression evaluates to the failed value
* the return type of the right most chain element is the type the entire expression always evaluates to
* condition: type must implement Tryable
* success variant?

const steht für eine Konstante deren Wert

nicht geändert werden kann und mut für einen Wert welcher Veränderbar ist.

**Enums**

* <enum\_name>::<enum\_variant>
* no overloading of enum variants, every enum variant in scope must be unambiguous
* `use <enum\_name>::<enum\_variant>` can bring a variant into scope
* plp p. 146
* C# => every use of a variant must be prefixed with a type name
* C, C++ => every variant visible in a scope must be distinct

**Function Overloading**

* forbidden,
* C++, Java, C#

**Operator Overloading**

**Generic Functions**

* plp p. 148 fff.
* generate specific functions for each combination of argument types
* https://youtu.be/Sg4U4r\_AgJU
* Notation makes a tremendous amount of difference. If you have a good notation, it makes it a lot easier to do a job. And a lot of language work is thinking about what is an appropriate notation for the kinds of things that you want to do in a particular programming language. Another issue, that is sort of related, is programmer efficiency. How long does it take a programmer to write a piece of code? Can you write it in 2 seconds or 20 minutes? That's a big difference and so if nothing else matters but the amount of time that
* the programmer takes to write it, then you want something that's very easy.

The future of programming languages

https://youtu.be/oMpqj\_nMsg0

register machine vs. stack machine

https://youtu.be/14CPI1qBd7o

Dont write a programming language

https://blog.dhsdevelopments.com/dont-write-a-programming-language

plp p. 10:

Donald Knuth has suggested that programming be regarded as the art of telling another human being what one wants the computer to do.

Rust Tips:

* inside of large generic functions add an
* inner non generic function to avoid generating duplicated code

plp => Programming Language Pragmatics 3rd Edition