Functional Programming

Functional Programming

- Functional programming is a **declarative** programming paradigm (**declarative** as opposed to imperative)
- Programs are created by **evaluating sequential functions** **rather than setting values** (recursion instead of for-loop)
- Use **pure functions** which predictably convert input into output without any dependencies on external **"side effects"** like shared variables, I/O, date/time, program state, etc.)
- Functions can be composed into **sequences**, passed as arguments, returned, saved
- The functional paradigm results in **highly modular code**

.. https://en.wikipedia.org/wiki/Functional_programming
.. https://www.educative.io/blog/what-is-functional-programming-python-js-java

```
;;; factorial in Lisp

(defun factorial (N)
  (if (= N 1)
      1
    (* N (factorial (- N 1)))))


(factorial 4).  ;;; ouput: 24
```

```
% factorial in Erlang

factorial(0) -> 1;
factorial(N) when N > 0 ->
    N * factorial(N-1).

factorial(4).  % 24
```

```
# imperative (prescribe steps, python):

def factorial(n):
    f=1
    for i in range(1,n+1):
        f = f*i
    return f

factorial(4) # 24
factorial(6) # 720


# --------------------------------
# functional (declare rules, python):

from functools import reduce

def multiply(x, y):
    return x * y

def factorial(n):
    return reduce(multiply, range(1,n+1))

factorial(4)  # 24
factorial(6)  # 720
```

```
-- Example: factorial function is a Haskell:
fac :: (Integral a) => a -> a
fac 0 = 1
fac n = n * fac (n - 1)

main :: IO ()
main = print $ fac 5
```

Example: Towers of Hanoi

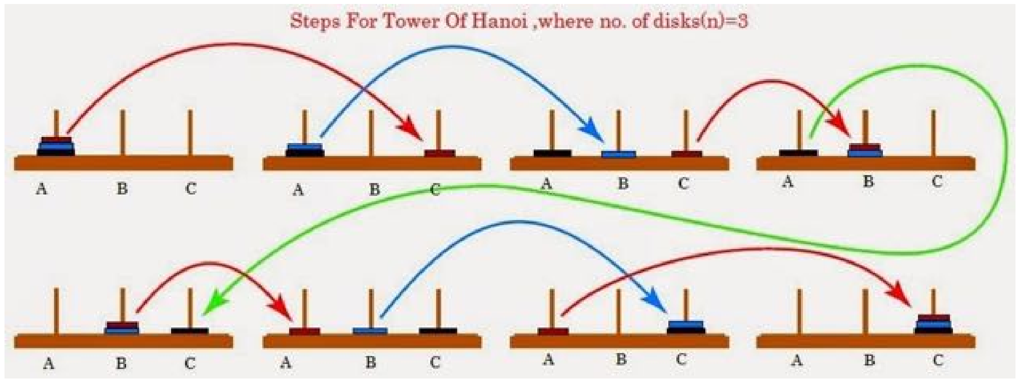
This problem is very difficult to program using imperative style, but it becomes easy and elegant when we use functional style of programming

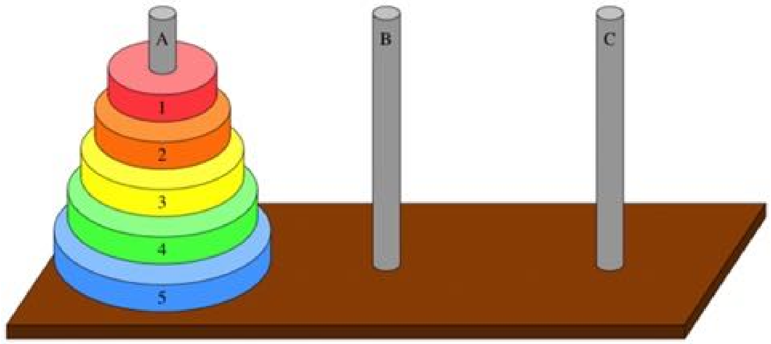
The game "Towers of Hanoi" uses three rods and a set of disks stacked on the first rod from largest disk on bottom to smallest on top – thus forming a conical tower.

The aim of the game is to move the tower of disks from one rod to another

- Only one disk can be moved at a time
- We can only take the top-most disk (can not take disk from the middle)
- The disk can NOT be placed on top of a smaller disk

https://en.wikipedia.org/wiki/Tower_of_Hanoi





```
def hanoi(n, source, helper, target):

    if n > 0:

        # move tower of size n - 1 from source to helper
        hanoi(n - 1, source, target, helper)

        # move disk from source peg to target peg
        if source:
            target.append(source.pop())

        # move tower of size n-1 from helper to target
        hanoi(n - 1, helper, source, target)


source = [4,3,2,1]
target = []
helper = []

hanoi(len(source),source,helper,target)
print(source, helper, target)
```

Success Story – Pugs (Perl6 & Haskell)

Pugs is a compiler and interpreter for **Perl6** (and now for **Raku** programming language).

2005 - **Audrey Tang** started successful project of using **Haskell** to create compiler for **Perl6**.

Haskell was successful where ohter languages have failed. Before 2005 there were several unsuccessful attempts of implementing the compiler in different languages, including **Perl5**. The problem was in the complexity of the **Perl6** syntax, which is context-sensitive (same expression may mean different things depending on its context).

""" Audrey Tang ... spent a month learning Haskell, and jumped from there to Pierce's book Types and Programming Languages (Pierce, 2002). The book suggests implementing a toy language as an exercise, so Tang picked Perl 6. At the time there were no implementations of Perl 6, at least partly because it is a ferociously difficult language to implement. Tang started her project on 1 February 2005. A year later there were 200 developers contributing to it; perhaps amazingly (considering this number) the compiler is only 18,000 lines of Haskell (including comments) (Tang, 2005). """
- https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/history.pdf

Pugs development is now placed on hold, most **Camelia, the Raku mascot** implementation efforts now taking place on **Rakudo** (**Rakudo** is a compiler for **Raku**)

- https://en.wikipedia.org/wiki/Raku_(programming_language)
- https://en.wikipedia.org/wiki/Pugs_(programming)
- https://en.wikipedia.org/wiki/Rakudo
- https://en.wikipedia.org/wiki/Audrey_Tang

Haskell

Camelia, the Raku mascot

Audrey Tang

```
# Raku code
my @array = 'a', 'b', 'c';
my $element = @array[1];    # $element equals 'b'
my @extract = @array[1];    # @extract equals ('b')
my @extract = @array[1, 2]; # @extract equals ('b', 'c')
```

Functional Programming Languages

Functional Programming Languages:

- **Lisp** (since 1958) – List Processor, many dialects (**Racket**, **Common Lisp**, **Scheme**, **Clojure**, etc.)
- **Emacs Lisp** (since 1985) - dialect of Lisp used in Emacs editor for implementing most of the editing functionality (the rest is in C)
- **Clojure** - a functional-first dialect of **Lisp** used on Java virtual machine (JVM)
- **Haskell** - clear **favorite language for functional programming** (tutorial - https://www.youtube.com/watch?v=02_H3LjqMr8)
- **Erlang** - best functional language for **concurrent systems** (tutorial - https://www.youtube.com/watch?v=IEhwc2q1zG4)
- **Elixir (descendent from Erlang)** - language for **concurrent systems** (tutorial - https://www.youtube.com/watch?v=pBNOavRoNL0)
- **Elm** - a functional language that compiles to JavaScript
- **Carp** - for interactive and performance sensitive use cases (games, sound synthesis, visualizations) - https://github.com/carp-lang/Carp

Languages with Functional Feaures:

- **F#** - functional, imperative, and OO
- **OCaml** – functional, imperative, and OO styles (was used for 1st compiler of **Rust**)
- **Rust** – supports FP, variables are immutable by default
- **Scala** – on top of Java, supports OOP and FP
- **JavaScript** - not functional-first, but uses **FP** due to its asynchronous nature
- **Python, PHP, C++** - multi-paradigm languages
- **Java** - mostly OOP

Three Styles of Programming

- Imperative Programming (IP)
- Object Oriented Programming (OOP)
- Functional Programming (FP).

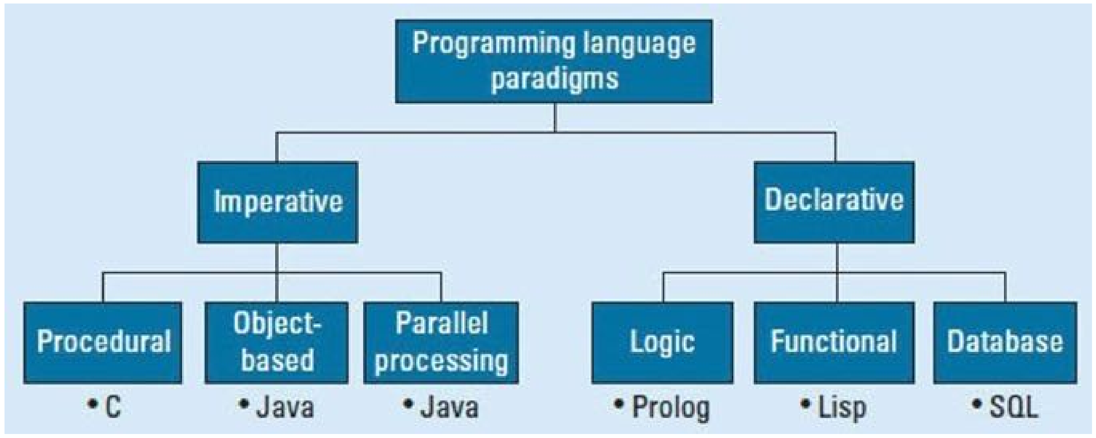
.. IP prescribes the order of operations to perform (for-loops)
.. FP declares what needs to be done, but actual execution is abstracted away (maps, function-composition, etc.)

.. OOP brings together data and associated methods.
.. FP says that data and behavior are distinctively different things and should be kept separate for clarity.

Real projects can benefit from using all three approaches as needed.

Also you need to take into account the maintenance, and how you expect the requirements to evolve.

Rigid OOP or FP both can be difficult to maintain or change.



Lambda Calculus

Invented in 1930 by Alonzo Church as method to describe algorithms

Each term "t" in lambda calculus is composed out of just three elements

```
t :==
      x       Variable
      λx.t    Abstraction  f(x) = t
      t  t    Application (first term – function)
```

Lambda calculus is Turing complete!

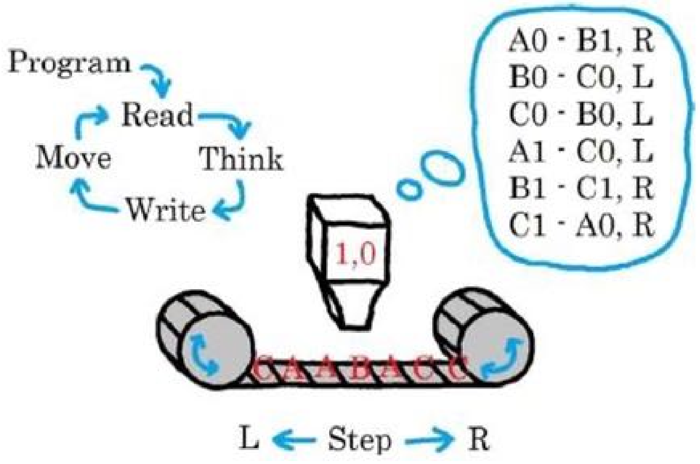
Although not very readable.

Here for example how you can define a function to calculate a factorial (https://math.stackexchange.com/questions/2221432/lambda-calculus-factorial):

$$! = \lambda k.\, k\left(\lambda p.\, p\left(\lambda abg.\, g\left(\lambda fx.\, f(afx)\right)\left(\lambda f.\, a(bf)\right)\right)\right)\left(\lambda g.\, g\,(\lambda h.\, h)\,(\lambda h.\, h)\right)\left(\lambda ab.\, b\right)$$

Alonzo Church

Turing Machine (1936)



A Turing machine is an abstract machine that manipulates symbols on a strip of tape according to a table of rules.
The machine "reads" the symbol, then writes and moves (based on the table of rules), then repeat.

Alan Turing

History of Functional Programming

The **lambda calculus**, developed in the 1930s by Alonzo Church.

lambda calculus is a formal system of computation built from function application.

In 1937 Alan Turing proved that the **lambda calculus** and Turing machines are equivalent models of computation, showing that the **lambda calculus** is **Turing complete**.

lambda calculus forms the basis of all functional programming languages.

.. https://en.wikipedia.org/wiki/Lambda_calculus
.. https://en.wikipedia.org/wiki/Lambda_calculus_definition

How lambda looks:

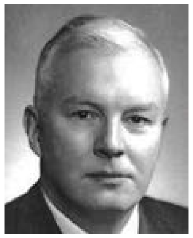
```
0 := λf.λx.x
1 := λf.λx.f x
2 := λf.λx.f (f x)
3 := λf.λx.f (f (f x))
```

Lisp (since 1958, **LISP** = LISt Processor) - uses syntax with parentheses. Today, the best-known general-purpose Lisp dialects are: **Racket, Common Lisp, Scheme, Clojure**
.. https://en.wikipedia.org/wiki/Lisp_(programming_language)

Haskell (since 1987) – named after logician Haskell Curry (1900-1982)
.. https://en.wikipedia.org/wiki/Haskell_(programming_language)

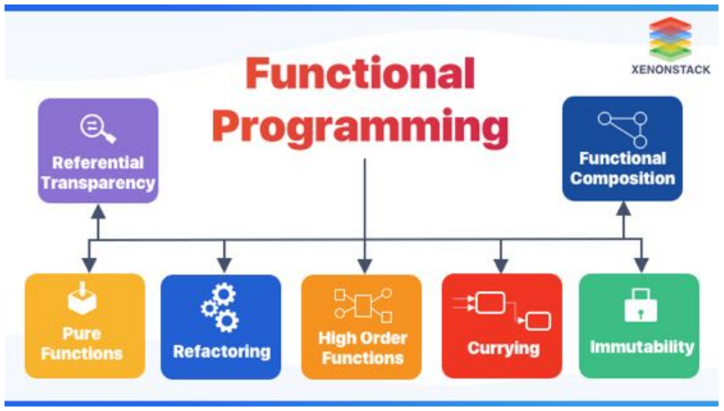
```
;;; factorial in Lisp

(defun factorial (N)
  "Compute the factorial of N."
  (if (= N 1)
      1
    (* N (factorial (- N 1)))))


(factorial 4)

ouput: 24
```

Haskell


Haskell Curry
(1900-1982)

Functional Programming ...

- https://www.educative.io/blog/what-is-functional-programming-python-js-java



Functional programming:

- **Pure functions** – convert input to output. The output depends only on the input, no side effects
- **Function composition** - execute many pure, single-purpose functions
- **Easy debugging** – because **pure functions** are predictable, easy to test
- **Modular** – **pure functions** naturally split code into steps/modules
- **Clean code** - easy to read
- **Referential transparency** - any function output can be replaced with its value
- **Lazy evaluation** – can reuse results from previous steps
- **Immutable variables** – help to make parallel programming reliable
- **No while- or for- loops.** Instead use map(), filter(), reduce(), etc.

- **Functional composition:**
  for example chaining functions:
  total_duration = events().extract_duration().compact().sum().round()
- **Currying:**
  transformation of a function with multiple arguments into a sequence of single-argument functions,
  for example converting **f(a, b, c, ...)** into a function like this **f(a)(b)(c)**
- **Recursion:**
  recursive function - calls itself, either directly or indirectly
- **First-class functions:**
  functions are treated like any other value (we can pass functions as parameters or store them as variables)
- **Higher Order functions:**
  can accept other functions as parameters or return functions as output
- **Monads:**
  overloading function composition to perform some **exrta computation on the intermediate value** (changing the type, logging, pass around a token, checking and processing the error, checking for missing values, ...)

Recursion

```
def myCount(i):
    if i >= 10:
        return 0
    return i + myCount(i+1)

myCount(1);
```

Currying

Break down a function that takes multiple arguments
into a series of functions that take part of the arguments.
Example - adding two numbers can be done as f(a,b) or f(a)(b).

```
def f(a):
    def g(b):
        return a+b
    return g

f(3)(4)  # 7
```

Closure

An inner function that has access to the outer (enclosing) function's variable

```
def make_multiplier_of(n):
    def multiplier(x):
        return x * n
    return multiplier

times3 = make_multiplier_of(3)
times5 = make_multiplier_of(5)

print(times3(9))          # 27
print(times5(3))          # 15
print(times5(times3(2)))  # 30
```

Lazy evaluation

Expressions are not evaluated when they are bound to variables, but their evaluation is deferred until their results are needed by other computations.

Immutability

You do not change values of variables. Variables are immutable.
You do not change state of anything. Everything is immutable.
You avoid mutable data and side-effects (dependence on global/external data, file or network IO).

In imperative language you can re-assign value

```
a = b;
a = c;
```

In functional language the same syntax would have different meaning:

Declare a variable «a» and assign «b» to it.
Declare another variable «a» and assign «c» to it.

You can have mutable variables, but the syntax changes, for example in F#:

```
let mutable a = b;
a <- c;
```

Immutability is not always the answer.
Imagine a huge Pandas DataFrame. You probably don't want to create and pass copies of this huge DataFrame between functions – you will run out of memory.

Immutability:

- makes it easier to write bug-free code (especially when you have many developers)
- makes it easier to debug the code
- makes it easier to write thread-safe code
- allows compiler to make effective optimizations (for speed - parallel execution, converting immutable into mutable, etc.)

For debugging a map if you want so see on which step you are - in advance convert your data to list of tuples [(d0,0),(d1,1), ...]

Side-Effects (to avoid) - modifying global/external variables, writing to a file, network, or screen, triggering any external process.

trie data structures (pronounced “tree”) - no property or subproperty can change (regardless of “depth”)

functor - something that can be mapped over.

stream - a list expressed over time

Monads - design pattern to chain operations/functions together.

Monad overloads function composition to perform some exrta computation on the intermediate value (changing type, checking for error or existance , ...).

Monads typically use two functions:

- **bind()** (>>= in Haskell) - to convert our functions to have composable signatures
- **unit()** - wrap the value in a basic container for function to consume

Monads apply a function that returns a wrapped value to a wrapped value.
Monads have a "**bind**" function "**>>=**" to do this.

Monads examples: Failure Monad, Error Monad, List Monad, Reader Monad, State & Writer Monad

Monads:

- **in pictures** - http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html
- **in 15 minutes** - http://nikgrozev.com/2013/12/10/monads-in-15-minutes/
- **in Javascript** - http://blog.klipse.tech/javascript/2016/08/31/monads-javascript.html

Mondad in Philosophy
The term monad (from Greek μονάς monas, "singularity" in turn from μόνος monos, "alone") ... refer to a most basic or original substance. As originally conceived the Pythagoreans, the Monad is the Supreme Being, divinity or the totality of all things. In the philosophy of Gottfried Wilhelm Leibniz, there are infinite monads, which are the basic and immaterial elementary particles, or simplest units, that make up the universe.
- https://en.wikipedia.org/wiki/Monad_(philosophy)

Monad in Category Theory
an endofunctor (a functor mapping a category to itself), together with two natural transformations required to fulfill certain coherence conditions
- https://en.wikipedia.org/wiki/Monad_(category_theory)

Monad IN Functional Programming
A monad is a software design pattern with a structure that combines program fragments (functions) and wraps their return values in a "monadic" type with **additional computation**.
Monads help to turn complicated sequences of functions into succinct pipelines that **abstract away control flow, and side-effects** (dealing with "real" world - errors, type mismatch, IO, database operations, etc. etc.).
Haskell’s I/O is based on Monads.

- https://en.wikipedia.org/wiki/Monad_(functional_programming)

Examples of good software that is written using functional programming today

- **Pandoc** - a Haskell library a9d tools for converting from one markup format to another - https://github.com/jgm/pandoc
- **Flow** – (facebook, OCaml) – a static typechecker for JavaScript - https://github.com/facebook/flow
- **Infer** (facebook, OCaml) - a static analysis tool for Java, C++, Objective-C, and C - https://github.com/facebook/infer
- **Reason** - write simple, fast and quality type safe code (JavaScript & OCaml) - https://github.com/reasonml/reason
- **MirageOS** (OCaml) - a library operating system - https://github.com/mirage/mirage
- **OCaml compiler** - https://github.com/ocaml/ocaml
- **Glasgow Haskell Compiler** - https://github.com/ghc/ghc
- in Erlang – **RabbitMQ, Amazon SimpleDB, Apache CouchDB**
- in Scala – Apache Spark, Apache Samza, Twitter Finagle, Apache Kafka
- in Clojure – **Apache Storm**

map-reduce:

- in functional programming
- in computer clusters (Google Big Table, Hadoop/Spark)

Function Composition, Monoid, Monad

```
x : a          where a = any type
f : a -> a
g : a -> a
```

In mathematics, **function composition** is an operation "∘" that takes two functions f and g, and produces a function h = g∘f such that

```
    (g∘f)(x) = g(f(x))
```

Associativity:

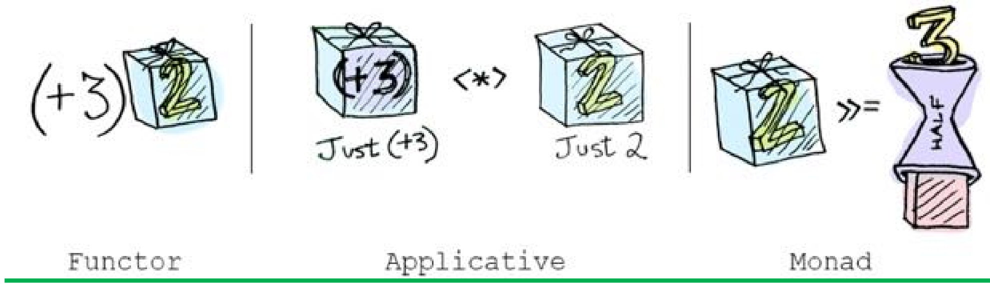
```
    h∘(g∘f)(x) = h(g(f(x))) = (h∘g)∘f(x)
```

- https://en.wikipedia.org/wiki/Function_composition
- https://en.wikipedia.org/wiki/Monoid

Monoids are semigroups with identity

- a set
- with an associative binary operation
- with an identity element

For example, the functions from a set into itself form a monoid with respect to function composition



\a   lambda of a
>>=   bind operator (shove operator)

f : a -> Ma
g: a -> Ma

\a -> [(fa) >>= \a -> (ga)]

- https://en.wikipedia.org/wiki/Monad_(functional_programming)

A **monad** is a functor.
A **monad** is a software design pattern with a structure that combines functions and wraps their return values in a "**monadic**" type **with additional computation**.
Monads define **two operators**:

- one to wrap a value in the monad type,
- and another to compose together functions that output values of the monad type (these are known as **monadic functions**)

Functional languages use **monads** to turn complicated sequences of functions into succinct pipelines that **abstract away control flow, and side-effects**.

Brian Beckman: Don't fear the Monad
 - https://www.youtube.com/watch?v=ZhuHCtR3xq8
Brian Beckman: The Zen of Stateless State - The State Monad
 - https://www.youtube.com/watch?v=XxzzJiXHOJs

Haskell Starter

https://riptutorial.com/haskell

```
--  file helloworld.hs  --

main :: IO ()
main = putStrLn "Hello, World!"


--  compile  --

ghc helloworld.hs


--  execute (3 ways)  --

./helloworld

runhaskell helloworld.hs

runghc helloworld.hs
```

Haskell is a compiled language

ghc = Glasgow Haskell Compiler
ghci = interactive (Prelude, run commands from prompt)

https://www.haskell.org

Install Haskel on Mac:

```
brew install ghc
```

Run interactively:

```
~ ››› ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help

Prelude> putStrLn "Hello World!"
Hello World!

Prelude> :quit
Leaving GHCi.

~ ›››
```

Haskell Tutorial by Derek Banas (75 minutes – Learn Haskell in One Video
- https://www.youtube.com/results?search_query=haskell+tutorial

Learn You a Haskell for Great Good!
- http://learnyouahaskell.com/chapters
- http://book.realworldhaskell.org/read/

User Guide:
- https://downloads.haskell.org/~ghc/7.4.1/docs/html/users_guide/index.html
- https://downloads.haskell.org/~ghc/9.0.1/docs/html/users_guide/ghci.html

Haskell – more ...

- Why Haskell is so darn fast? (only 5-10% slower than C)
  Because it is compiled, not interpreted. No virtual machine.
  Purity. Static types. Laziness. But above all, being sufficiently high-level that the compiler can radically change the implementation without breaking your code's expectations.
  - https://stackoverflow.com/questions/35027952/why-is-haskell-ghc-so-darn-fast

- How to generate random numbers in Haskell?
  - System.Random module
  - http://learnyouahaskell.com/input-and-output#randomness
  - https://hackage.haskell.org/package/random

- How you handle database connections in Haskell ?
  - https://hackage.haskell.org/package/HDBC
  - https://hackage.haskell.org/package/mysql-simple
  - https://github.com/paul-rouse/mysql-simple/blob/master/test/main.hs

- How do we process big DataFrames in Haskell ?
  - https://hackage.haskell.org/package/Frames

- How to make a server in Haskell ?
  - https://catonmat.net/simple-haskell-tcp-server

- Husk - a functional golfing language, inspired by (and implemented in) Haskell
  - https://github.com/barbuz/Husk

Haskell vs production environment

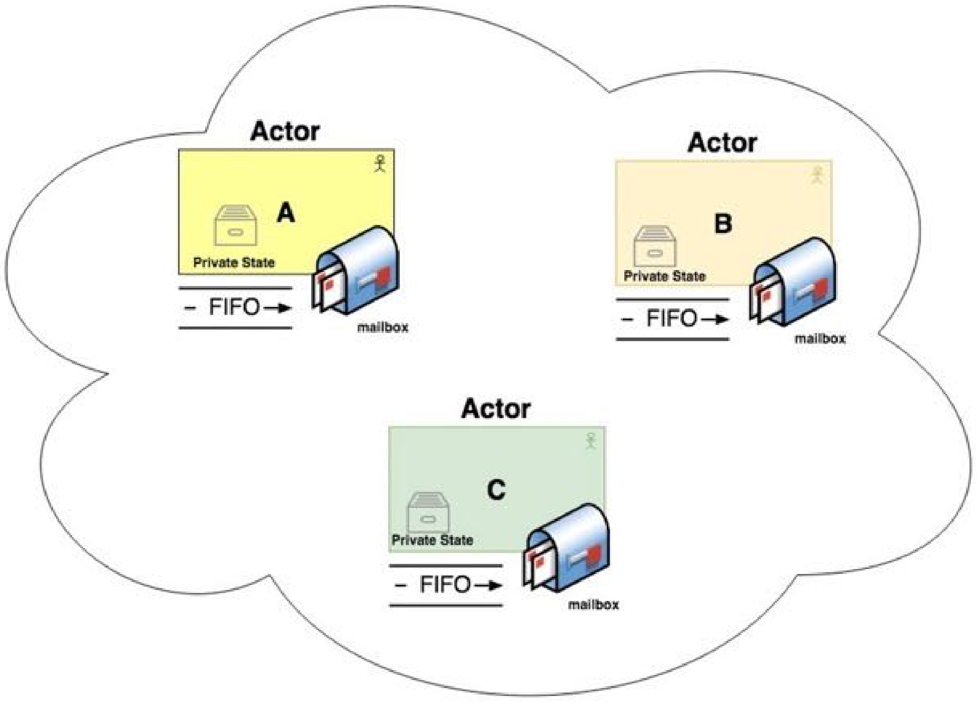
Haskell community is "academic", interested in developing cool ideas,
but unfortunately not good at providing solid bug-free tools for production environment.

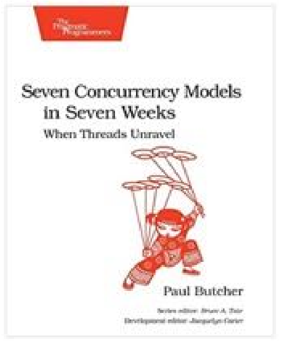
Sometimes the compiled code is fast, sometimes it is amazingly slow. Sometimes a minor change in code can make it slow.

Actor Model

Concurrency without any shared memory.
Data sent between actors via messages.

- https://finematics.com/actor-model-explained/
- https://www.youtube.com/watch?v=ELwEdb_pD0k
- also search for **Vagif Abilov**





Seven Concurrency Models in Seven Weeks: When Threads Unravel (The Pragmatic Programmers) - by Paul Butcher
https://www.amazon.com/gp/product/1937785653/

Functional Programming in Rust

According to the Stack Overflow Developer Survey 2021 conducted among over 80,000 developers, **Rust is the most beloved programming language**. And it won the title **for the sixth year running**.

Graydon Hoare

- https://www.linkedin.com/in/graydon-h-881374212/
- https://github.com/graydon
- 2001-2005 – Red Hat
- 2006-2014 - Mozilla
- 2016-2018 - Apple
- 2019 – present - Stellar

- Main benefit of Rust – makes code more reliable. Prevents segfaults and guarantees thread safety. Super fast. Open source
- The language grew out of a personal project begun in 2006 by Mozilla employee **Graydon Hoare**
- Hoare has stated that the project was possibly named after **rust fungi** (a plant disease "over-engineered for survival") and that the name is also a subsequence of "**robust**". https://www.reddit.com/r/rust/comments/27jvdt/internet_archaeology_the_definitive_endall_source/
- Mozilla began sponsoring the project in 2009. Original compiler was written in **OCaml**, then it was shifted into an LLVM-based self-hosting compiler written in Rust (2011). The first stable release, was released in 2015

Functional Rust:

- Pure functions – easy to write in Rust
- Mutability - Variables are immutable by default in Rust. Rust has strong control over mutability.
- Recursion - You can do recursion. You can also use iterators.
- Lazy evaluation (iterators).
- Type inference, compiler enforced. Type annotation on function params and return values are required. Generics. Type variables. Traits (interfaces).
- Function declarations in Rust are statements – they can not be assigned to variables or used as arbitrary value. But we can bind the function to a variable. And we can use **anonymous functions (closures)**, their declarations can be stored in variables. Closures don't need annotations.
- Higher order functions (hof) – composing functions together (runs fast). Example: take numbers 1..100, multiply by 2 and sum them together

```
fn hof_example() {
    println!("{}", (1..101).map(|x| x * 2).fold(0, |x, y| x+y));
}
```

- Declarative style is possible
- Currying - careful, may interfere with memory safety when multithreading

Rust ...

Closure - anonymous function that can be put into a variable and passed around

```
let add_one = |x| {
    // body of the function
    1 + x
};

println!("5 + 1 = {}.", 
add_one(5));
```

Currying - a way to reduce the number of params passed to a function. Currying means to use a function that gets a parameter and returns a function (lambda) that contains the parameter from before.

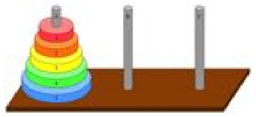
```
fn add(a: u32, b: u32) -> u32 {
    a + b
}

let add5 = move |x| add(5, x);

println!(add5(2));   // 7
```

Binding – a variable to a function

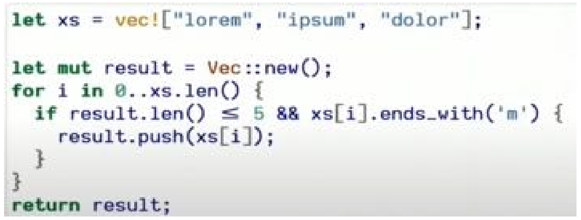
```
fn sum2(n1: i32, n2: i32) -> i32 {
    n1 + n2
}

let mysum = sum2;
println!("6+4={}", mysum(6,4));
```

Towers of Hanoi

```
fn move_(n: i32, from: i32, to: i32, via: i32) {
  if n > 0 {
      move_(n - 1, from, via, to);
      println!("Move disk from pole {} to pole {}", from, to);
      move_(n - 1, via, to, from);
  }
}

fn main() {
  move_(4, 1,2,3);
}
```

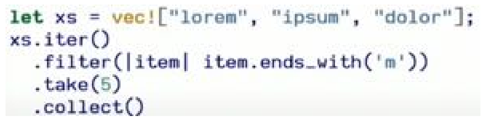


Imperative



```
let xs = vec!["lorem", "ipsum", "dolor"];

let mut result = Vec::new();
for i in 0..xs.len() {
  if result.len() ≤ 5 && xs[i].ends_with('m') {
    result.push(xs[i]);
  }
}
return result;
```

Declarative



```
let xs = vec!["lorem", "ipsum", "dolor"];
xs.iter()
  .filter(|item| item.ends_with('m'))
  .take(5)
  .collect()
```

- YOW! Lambda Jam 2019 - Amit Dev - Introduction to Functional Programming in Rust
  - https://www.youtube.com/watch?v=9x7W3_KKKeA
- Jeffrey Olson, "Functional Programming in Rust" 2021 (time 52:23 ... 1:00:30)
  - https://www.youtube.com/watch?v=CSk_QRE7GKg
- Michael Snoyman: From Haskell to Rust?
  - https://www.youtube.com/watch?v=HKXmEFvsi6M
- A Firehose of Rust, for busy people who know some C++
  - https://www.youtube.com/watch?v=IPmRDS0OSxM
- Why would a python programmer learn rust when there are no jobs in it (2019)
  - https://www.youtube.com/watch?v=IYLf8lUqR40
- Rust Tutorial (47 min) by Derek Banas (2016)
  - https://www.youtube.com/watch?v=U1EFgCNLDB8
- Lambda World 2018 - Rust and Haskell, sitting in a tree - Lisa Passing
  - https://www.youtube.com/watch?v=em6BOXY9jMY
- Rust: Functional Programming
  - https://www.youtube.com/watch?v=MjwAxZIMYDs
- Play with Rust online:
  - https://play.rust-lang.org

Rust for Machine Learning

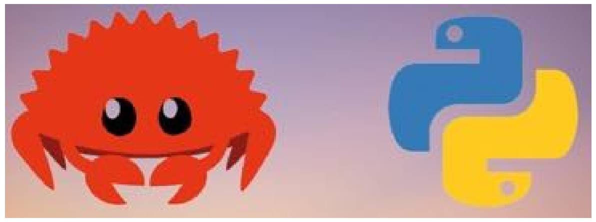
Rust can be used for Data Science:
- https://qvault.io/python/rust-vs-python/
- https://www.arewelearningyet.com/
- https://smartcorelib.org/user_guide/supervised.html

Linfa – ML Framework:
Linfa aims to provide a comprehensive toolkit to build Machine Learning applications with Rust.
Kin in spirit to Python's **scikit-learn**, it focuses on common preprocessing tasks
and classical ML algorithms for your everyday ML tasks.
- https://rustrepo.com/repo/rust-ml-linfa-rust-machine-learning

Tutorial:
- https://github.com/Steboss/ML_and_Rust

part 1: https://levelup.gitconnected.com/machine-learning-and-rust-part-1-getting-started-745885771bc2
part 2: https://levelup.gitconnected.com/machine-learning-and-rust-part-2-linear-regression-d3b820ed28f9
part 3: https://levelup.gitconnected.com/machine-learning-and-rust-part-3-smartcore-dataframe-and-linear-regression-10451fdc2e60

Rust doesn't have convenient Data Science environments similar to Python Jupyter or R-Studio.
It doesn't have rich ML libraries like Python or R.
But Rust can be effectively used to process data and prepare it for ML or Analytics.



You can invoke Rust executables as external processes.
Or you can call Rust libraries from inside Python code:
- http://saidvandeklundert.net/learn/2021-11-06-calling-rust-from-python/
- http://saidvandeklundert.net/learn/2021-11-18-calling-rust-from-python-using-pyo3/
- https://python.plainenglish.io/using-python-in-rus-and-trust-in-python-ac5cf77d5ece