Marek Milkovič

Programming Languages & FFI (Foreign Function Interface)

April 2026

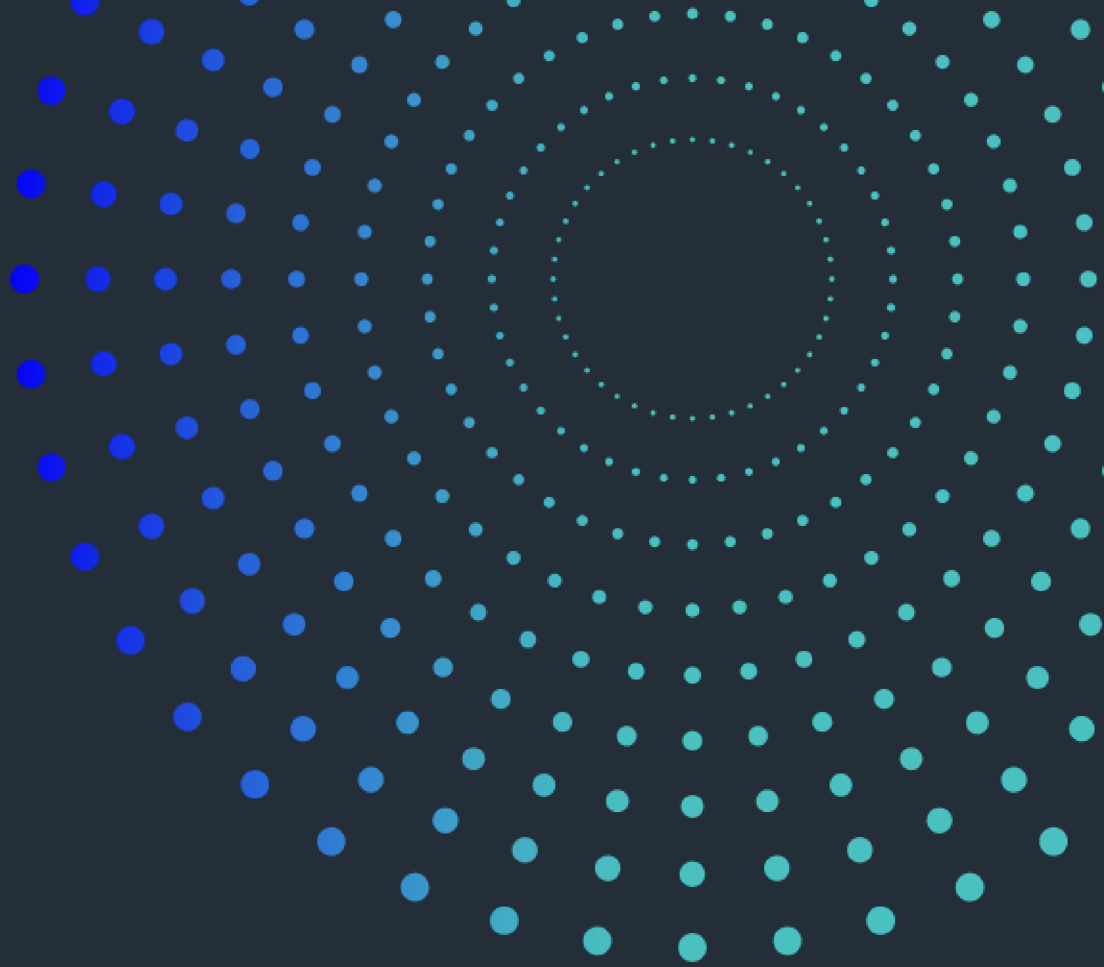
Copyright © 2025 Gen Digital Inc. All rights reserved.

Who am I?

- Marek Milkovič
- Engineering Manager @ Gen
  - Also known as Avast/AVG
  - Still very hands-on with software engineering
- Software engineer before that for ~10 years
- Former student of FIT

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages



Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

- Language used to "talk" to computers
- Language with well-defined syntax and semantics used to write instructions for computer to execute
- Execution usually needs another translation program – compiler, interpreter, ...
- Many different languages with many different ways to categorize them
  - Level of abstraction
  - Programming paradigms
  - Typing system
  - Translation
  - Memory management

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Level of abstraction

- Distinction based on how close is the language to mapping to hardware concepts
- **Low level**
  - Provides at most basic abstractions but otherwise closely mapped to HW
  - Usually none to minimal runtime
  - Assembly, C, C++ (depends)
- **High level**
  - Higher level of abstractions decoupled from HW concepts
  - Heavy runtime or runtime directly provides the execution
  - Usually very portable
  - Python, Java, C#, SQL

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Programming paradigms

- Defines a way to structure a program, what are the basic building blocks and how to express & use them
- **Imperative**
  - **Procedural**
  - **Object-oriented**
  - **...**
- **Declarative**
  - **Functional**
  - **Logic**
  - **...**

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Imperative vs Declarative

- **Imperative**
  - Defines "how" to solve to problem
  - Program written as series of steps which change the state
  - Usual building blocks – variables, loops, conditions
- **Declarative**
  - Defines "what" needs to be solved (or what is the desired state)
  - How to solve it is left up to the runtime

```
output = []
for n in seq:
    if n > 10:
        output.append(n * n)
```

```
SELECT n*n
FROM seq
WHERE n > 10
```

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Imperative - Procedural

- Program is split into smaller units – procedures/functions
- Values are shared across the procedures in parameters
  - Alternatively via global state
- C, Go

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Imperative – Object-oriented

- Program defined as a collection of objects communicating between each other
- Object encloses data + behavior under the same abstraction
- **Class-based OOP**
  - **Classes** – template for creating objects
    - Object is an instance of a class
  - 3 main pillars
    - **Encapsulation** – state of an object accessible exclusively via interface (methods)
    - **Inheritance** – class can extend other classes and create class hiearchies
    - **Polymorphism –** different objects might act differently but both can share the same interface
  - C++, C#, Java, Python
- **Prototype-based OOP**
  - Objects created directly from other objects (creating prototype chains)
  - JavaScript

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Declarative - functional

- Program expressed using (mathematical) functions which are doing the computation
- Roots in lambda calculus
- Function is treated as any other value – can be passed around
- Based on principles
  - Immutability – data cannot be changed once created
  - Pure functions – no side-effects, same input = same output
- Haskell, LISP, F#

```
mySum :: [Int] -> Int
mySum []     = 0
mySum (x:xs) = x + mySum xs
```

```
total = foldl (+) 0 [1, 2, 3, 4]
ftotal = foldl (+) 0
x = ftotal [1, 2, 3, 4]
```

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Declarative - logic

- Program built from facts expressed in formal logic
- Result is a proof
- Prolog

```
parent(alice, bob).
parent(adam, bob).
parent(claire, alice).

grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```


Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Concurrent

- Program built from processes communicating between each other
- Not necessarily parallel
- Erlang, Elixir, Go (channels)

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Metaprogramming

- Program modifies itself (usually) during compile-time
- Generates additional code
- Template metaprogramming (C++)
- Python decorators, Java annotations, Rust macros, …

```
template<int N> struct Factorial {
    static constexpr int value = N * Factorial<N - 1>::value;
};

template<> struct Factorial<0> {
    static constexpr int value = 1;
};

static_assert(Factorial<5>::value == 120);
```

```
macro_rules! string_enum {
    ($name:ident { $($variant:ident),* $(,)? }) => {
        #[derive(Debug, Clone, Copy, PartialEq)]
        enum $name { $($variant),* }
        impl $name {
            fn as_str(&self) -> &'static str {
                match self { $(Self::$variant =>
                          stringify!($variant)),* }
            }
            fn from_str(s: &str) -> Option<Self> {
                match s { $(stringify!($variant) =>
                         Some(Self::$variant),)* _ => None }
            }
        }
    };
}

string_enum!(Color { Red, Green, Blue });
```

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Multi-paradigm

- Languages are usually multi-paradigm
- Increased interest in functional aspects even in imperative languages
  - Pure functions
  - High-order functions
  - Monadic types
- Decrease of interest in "OOP by the book"
  - Less inheritance hierarchies
  - Some design patterns can overengineer the code with unnecessary boilerplate
  - Modern languages decided to drop OOP – Rust, Go

```
input.parse::<i32>()
            .ok()
            .filter(|&n| n > 0)
            .map(|n| n * n)
            .and_then(|n| {
                if n < 1000 { Some(n) }
                else { None }
            })
            .map(|n| format!("result = {n}"))
```

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Typing system

- **Static vs Dynamic**
  - **Static** – types of variables needs to be known ahead of time, type is associated with the variable
  - **Dynamic** – types of variables determined during runtime, type is associated with the value
    - Usually types are treated the same way as any other value
  - Nowadays (some) dynamic languages are leaning towards static-like features
    - Python type hints
    - TypeScript
- **Weak vs Strong**
  - Weak – implicit conversions of types according to the operations on the values
  - Strong – no implicit conversions, the exact types enforced

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Memory management

- **Manual**
  - Author of the code explicitly requests and releases memory chunks
  - Predictable but very error-prone
- **Automatic**
  - The memory is managed automatically
  - Multiple implementations
    - **Reference counting** – obtaining reference to object increments reference count, release decrements, object is freed with counter reaching 0 (Swift)
    - **Garbage collector** – sweeps the memory on certain triggers (time, # of memory allocations, …) and removes unreachable objects (Java, Go)
    - **Compile-time ownership** – owner known at compile-time, when it goes out of scope the value is released too (Rust, C++ RAII)

Copyright © 2025 Gen Digital Inc. All rights reserved.

Programming Languages

Translation

- **Compiled**
  - Compiler which translates the code to the machine code
  - Fastest
  - C, C++, Rust
- **Interpreted**
  - Parsed and compiled as the code is executed
  - The code that haven't been hit is not translated
  - JIT (Just-in-Time) compilation possible
  - Slowest
- **Hybrid**
  - Compiled into bytecode / intermediate representation
  - Needs additional runtime (interpreter) to execute the bytecode (VM)
  - Java, C#

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

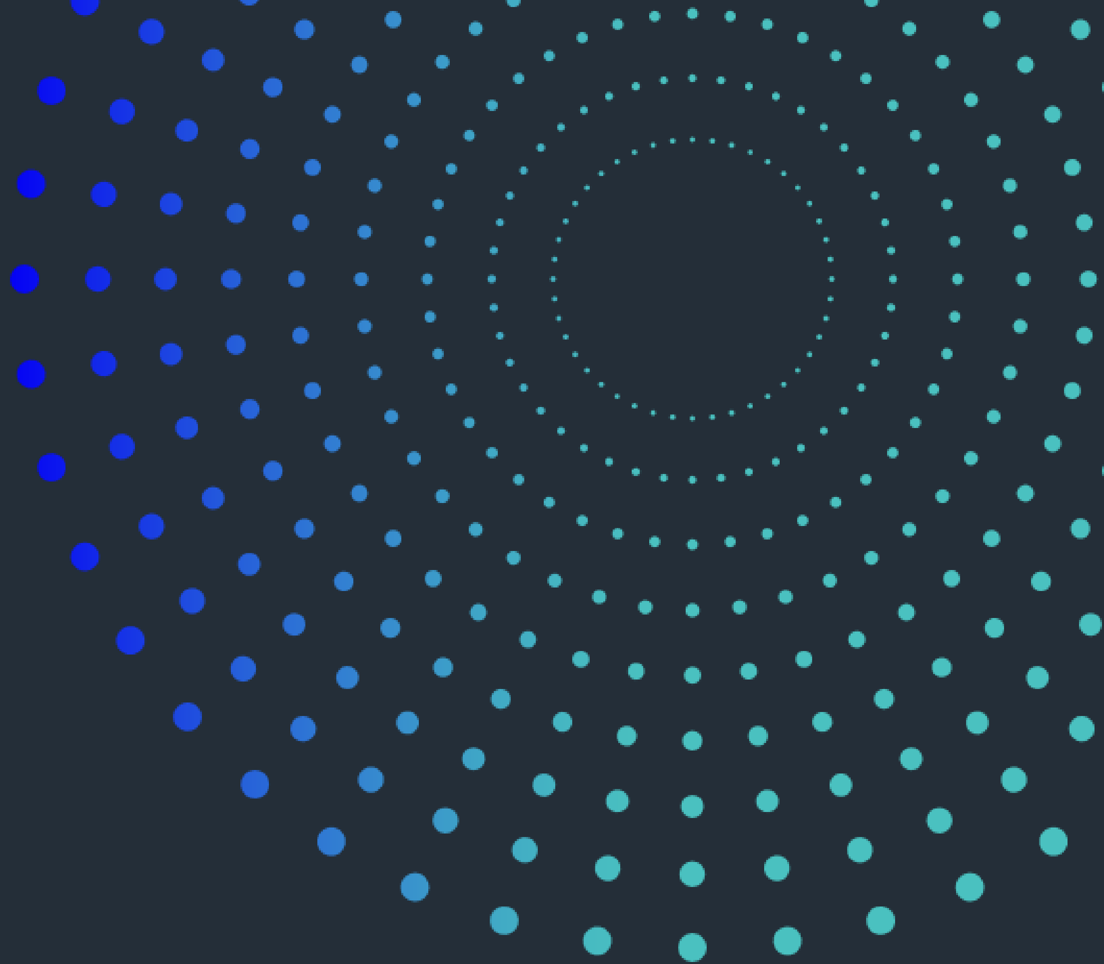
Programming Languages

Choosing a language

- Several questions to ask yourself
  - What am I building? Web server? Web page? CLI? Long running daemon?
  - What are the performance requirements? Do I need to handle 100k requests per second or 10 per hour?
  - What platform am I targeting? Web? Desktop? Mobile?
  - What's the domain? Am I training AI model? Am I communicating over network with my fridge?
  - What's the scope? Is it a single purpose one-shot job? Will it be huge project with 25 features?
  - What if I run into roadblocks? What's the community like? Are my chosen languages popular?
  - What do I like?
- Pick whatever, just...
  - Adjust it to your needs – do not get filtered just on picking a right language
  - Your first PoC doesn't have to be in "the final language" – just build something
  - Be prepared to explain yourself, why you picked the way you did (especially if it's something exotic)

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI



Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

- Foreign Function Interface
- Ability of a program written in one language to call functions or access data within program written in a different language
- Why even do this?
  - 3rd party library available only in a different language
  - Performance gain from using low-level language from higher level one
  - Plugin system / modding support for a software
  - Access to OS resources

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

Compilation

- **Compiler** translates into compilation units
- **Linker** takes compilation units and resolves them into a final product
  - **Executable file** – binary file mean to be executed on its own with its own entrypoint (.exe)
  - **Library –** binary file meant to be used by other software
    - **Static –** meant to bundled into other dynamic libraries / executable files and resolved during link time (.a, .lib)
    - **Dynamic –** meant to be used as-is and resolved during startup time / runtime (.so, .dylib, .dll)

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

API & ABI

- **API –** Application Programming Interface
  - What is the function called?
  - How many parameters does it accept?
  - What is the type of the parameters?
  - Defines the interface for software components to communicate on the source level
- **ABI –** Application Binary Interface
  - What is the memory layout?
  - What is the calling convention?
  - What are the symbols named as?
  - Defines the interface on binary level for compiled code to interact with system and each other

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

ABI - Calling convention

- Arguments passing via registers / stack
- Ordering of arguments (left-to-right, right-to-left)
- Who is supposed to cleanup the arguments (caller / callee)
- Location of return value
- x86 – cdecl, Pascal, fastcall
- x86-64 – microsoft x64, amd64

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

ABI - Data representation

- Each library (side) of FFI need to understand the differences between various data representations
- Structure memory layout
  - Aligned memory access is cheaper
  - Reordering or padding
- Size of primitive types
  - short, int, long
- Storage of arrays / strings

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

ABI

- Beware of ownership
  - What if C library calls malloc(), who should call free()?
  - We might be passing something that gets GC'd into C
- Error handling
  - Exceptions don't really work – rely on other mechanisms
  - Error codes, return values
- Parallelism
  - Is the underlying interface thread-safe?
  - Holding locks + crashes
  - Python-specific - GIL

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

How to solve it?

- **Rewrite**
  - Costly and time-consuming
  - Might not be viable in all situations
- **Use dynamic library and link it / load it during runtime**
  - Same ABI needs to be ensured
- **Compile to common target**
  - WebAssembly
  - Requires additional runtime, can hurt performance
- **Use RPC / IPC**
  - Costly and time-consuming
  - High overhead

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

C ABI

- C ABI is *lingua franca* of FFI world
- Stable & simple interface supported on each OS
- Many languages can interact with it
- May impose limits on the program
  - C++ functions with C ABI export can't use overloading

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

FFI

Demo

- Demo 1: C + Rust
- Demo 2: C + Python (ctypes)
- Demo 3: C + Python (Python API)
- Demo 4: Rust + Python (PyO3)

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Cooperation with FIT BUT

- Long running cooperation between GenDigital and FIT BUT
- Areas in which we offer cooperation
  - Development of systems for classification of threats
  - Detection of anomalies of malicious campaigns
  - Reverse engineering (required at least some prior knowledge)
  - AI / Agentic security
  - Other topics after mutual agreement
- Bachelor / master theses or summer internships
- Contact point: Daniel Snášel – isnasel@fit.vut.cz

Gen

Copyright © 2025 Gen Digital Inc. All rights reserved.

Thank you

Marek Milkovič

Engineering Manager

marek.milkovic@gendigital.com

Copyright © 2025 Gen Digital Inc. All rights reserved.

Gen™

GenDigital.com

Copyright © 2025 Gen Digital Inc. All rights reserved.