



LIGOLANG
A MARIGOLD PROJECT

Writing a Simple Smart Contract

Suzanne Soy
ligo@suzanne.soy

ligolang.org

Why Tezos?

Proof of Stake

Consensus-driven protocol upgrades

Formal methods

Ecosystem redundancy

Why Tezos?

Proof of Stake

i.e. no palm trees at the North Pole

Consensus-driven protocol upgrades

i.e. no hard forks

Formal methods

i.e. fewer bugs

Ecosystem redundancy

i.e. don't put all your Tez in the same basket

Why LIGO?

Is Michelson suitable for application developers?

```
parameter unit ;  
storage string ;  
code {  
  DROP ;  
  PUSH string "Hello" ;  
  NIL operation ;  
  PAIR  
}
```

Why LIGO?

Michelson is not very suitable for application developers!

Goal?

Reduce the barrier of entry and minimize risks of bugs

Why LIGO?

Michelson is not very suitable for application developers!

Goal?

Reduce the barrier of entry and minimize risks of bugs

Solution?

High-level language that compiles to Michelson with four syntaxes, to cater for different developers' taste.

LIGO Foundations

LIGO Foundation :: Syntaxes

Each syntax is as close as possible to the language it is inspired from

PascaLIGO a Pascal inspired syntax which provides an imperative developer experience.

Imperative paradigm describes the operations in sequences of instructions executed to change the program's state.

LIGO Foundation :: Syntaxes

Each syntax is as close as possible to the language it is inspired from

PascaLIGO a Pascal inspired syntax which provides an imperative developer experience.

CameLIGO an OCaml inspired syntax that allows you to write in a functional style.

Imperative paradigm describes the operations in sequences of instructions executed to change the program's state.

Functional paradigm considers computation as an evaluation of mathematical functions

LIGO Foundation :: Syntaxes

Each syntax is as close as possible to the language it is inspired from

PascaLIGO a Pascal inspired syntax which provides an imperative developer experience.

CameLIGO an OCaml inspired syntax that allows you to write in a functional style.

ReasonLIGO a ReasonML inspired syntax that allows you to write in a functional style.

Imperative paradigm describes the operations in sequences of instructions executed to change the program's state.

Functional paradigm considers computation as an evaluation of mathematical functions

LIGO Foundation :: Syntaxes

Each syntax is as close as possible to the language it is inspired from

PascaLIGO a Pascal inspired syntax which provides an imperative developer experience.

CameLIGO an OCaml inspired syntax that allows you to write in a functional style.

Formal methods

ReasonLIGO a ReasonML inspired syntax that allows you to write in a functional style.

JsLIGO a Javascript inspired syntax which provides an imperative developer experience.

Imperative paradigm describes the operations in sequences of instructions executed to change the program's state.

Functional paradigm considers computation as an evaluation of mathematical functions

LIGO Foundation :: Syntaxes

Each syntax is as close as possible to the language it is inspired from

PascaLIGO a Pascal inspired syntax which provides an imperative developer experience.

CameLIGO an OCaml inspired syntax that allows you to write in a functional style.

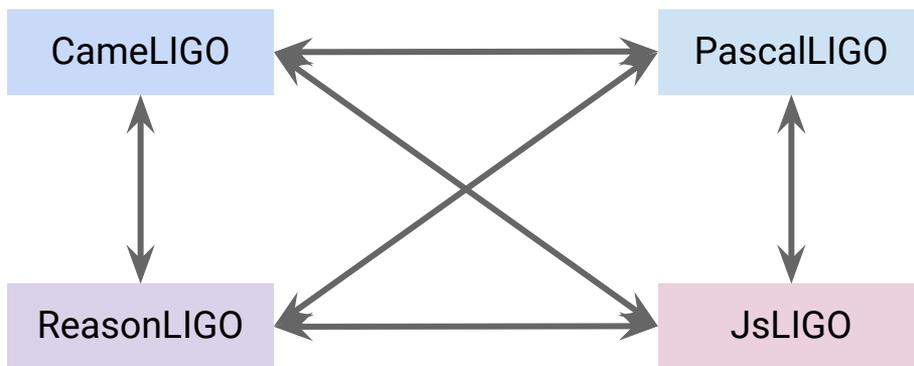
ReasonLIGO a ReasonML inspired syntax that allows you to write in a functional style.

JsLIGO a Javascript inspired syntax which provides an imperative developer experience.

Imperative paradigm describes the operations in sequences of instructions executed to change the program's state.

Functional paradigm considers computation as an evaluation of mathematical functions

LIGO Foundation :: Interoperability



LIGO Foundation :: Compiler Overview

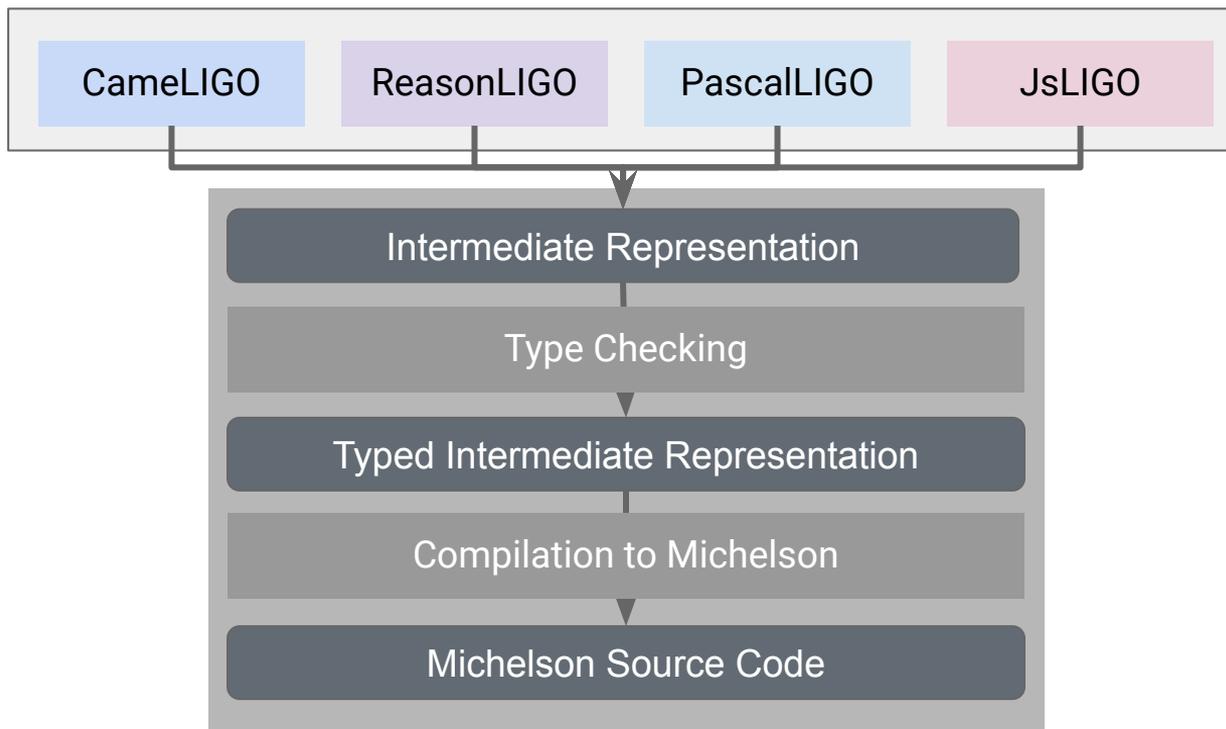
CameLIGO

ReasonLIGO

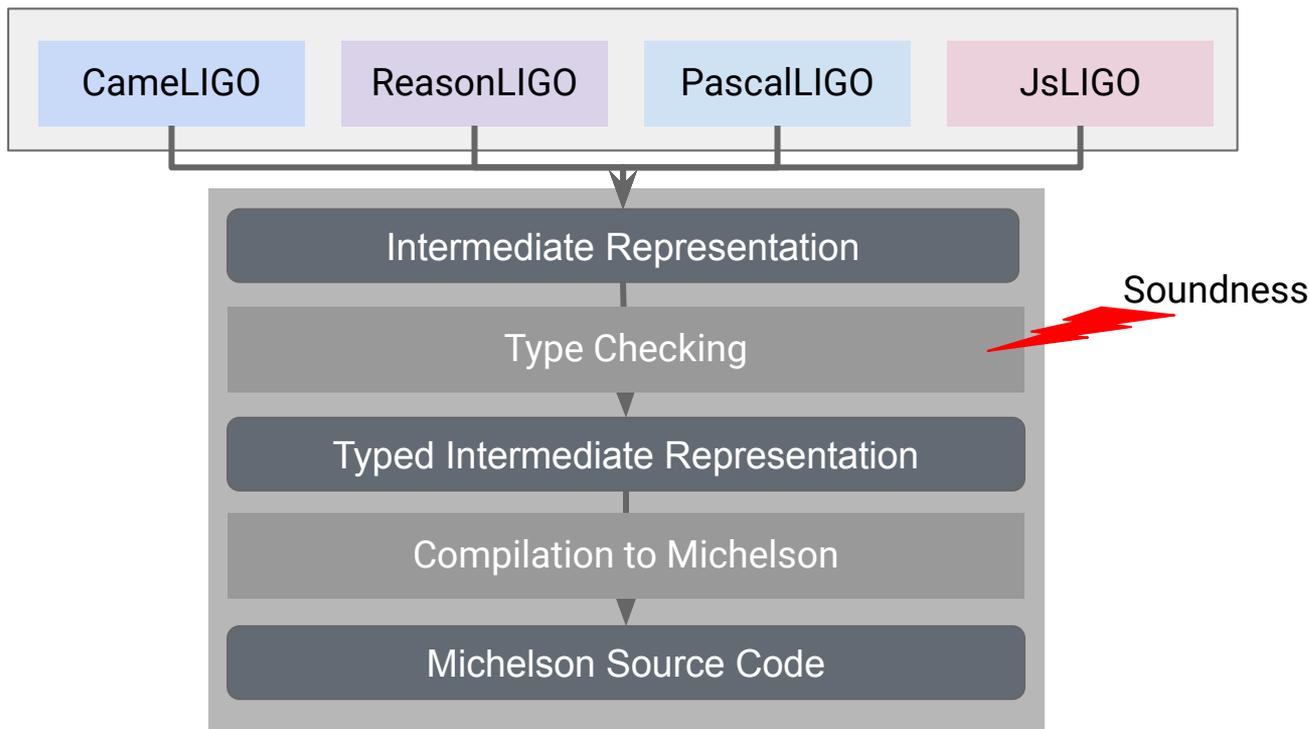
PascalLIGO

JsLIGO

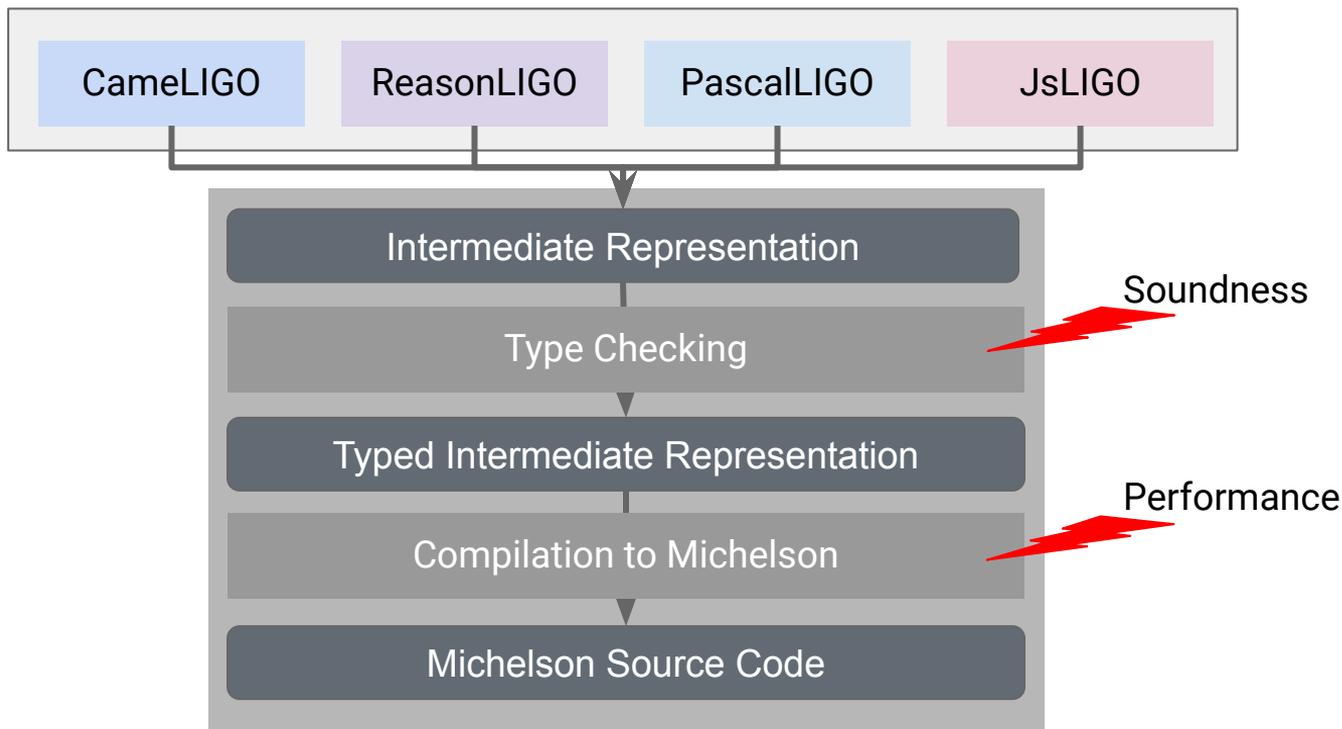
LIGO Foundation :: Compiler Overview



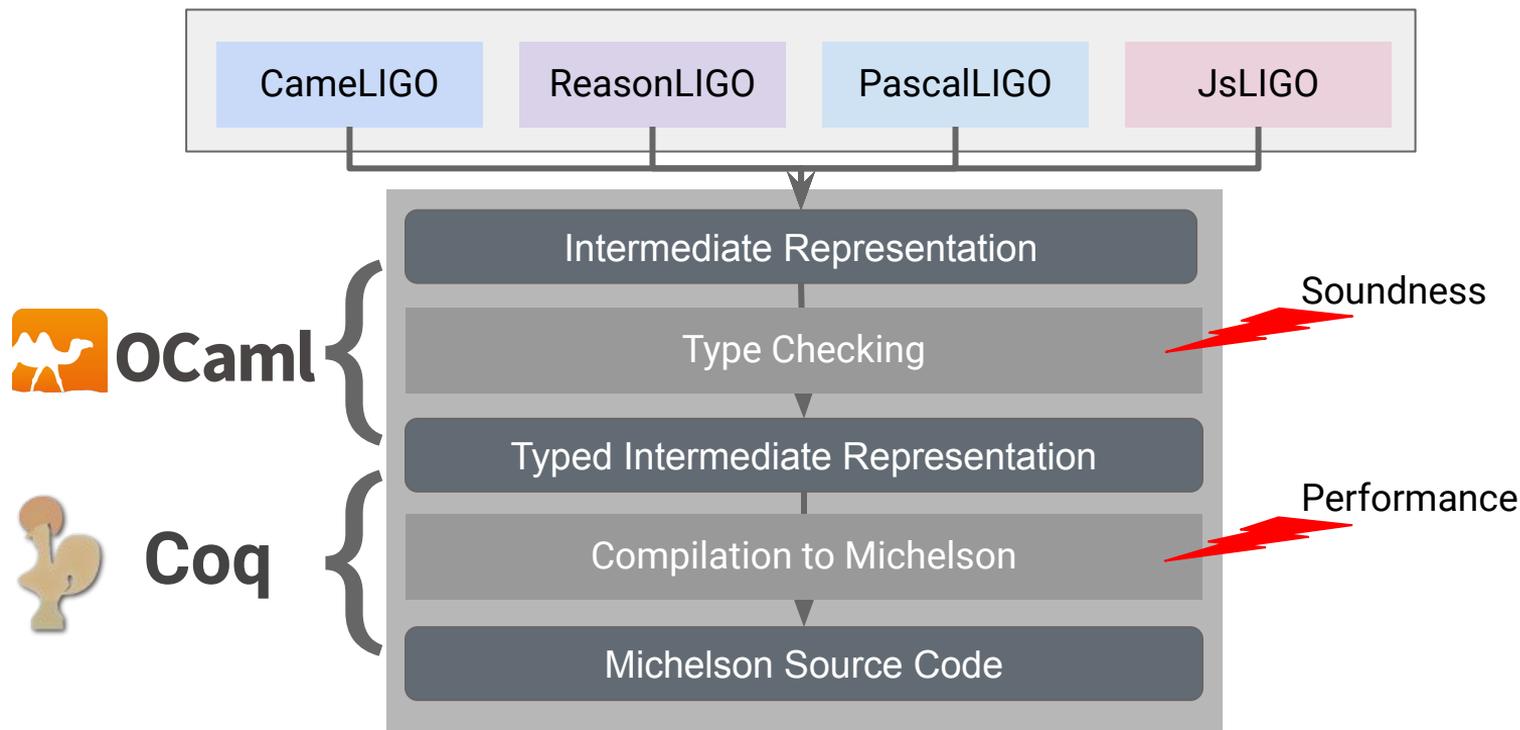
LIGO Foundation :: Compiler Overview



LIGO Foundation :: Compiler Overview



LIGO Foundation :: Compiler Overview



Contract Anatomy

Contract Anatomy :: Storage

```
type storage = int;
```

Storage

Contract Anatomy :: Parameter

```
type storage = int;  
type parameter =  
  | ["Increment", int]  
  | ["Decrement", int]  
  | ["Reset"];
```

Parameter

Storage

Contract Anatomy :: Operations

```
type storage = int;  
type parameter =  
  ["Increment", int]  
| ["Decrement", int]  
| ["Reset"];  
type result_ = [list<operation>, storage]
```

Parameter

Operations

Storage

Contract Anatomy :: Behavior

```
type storage = int;  
type parameter =  
  ["Increment", int]  
| ["Decrement", int]  
| ["Reset"];  
type result_ = [list<operation>, storage]
```

```
let main = ([action, store]: [parameter, storage]) : result_ => {  
  return [  
    list([]) as list<operation>,  
    match(action, {  
      Increment: (n: int) => store + n,  
      Decrement: (n: int) => store - n,  
      Reset:      ()      => 0  
    })  
  ];  
};
```



JsLIGO Overview

JsLIGO Overview :: Types

Native types

int, nat, **tez**, string, bytes...

// 1 "Hello" ...

JsLIGO Overview :: Types

Native types

int, nat, **tez**, string, bytes...

// 1 "Hello" ...

Tuples

[int, string, tez]

// (1, "Hello", 42 as tez)

JsLIGO Overview :: Types

Native types

int, nat, **tez**, string, bytes...

// 1 "Hello" ...

Tuples

[int, string, tez]

// (1, "Hello", 42 as tez)

Records

{ name: string }

// { name: John }

JsLIGO Overview :: Types

Native types	int, nat, tez , string, bytes...	// 1 "Hello" ...
Tuples	[int, string, tez]	// (1, "Hello", 42 as tez)
Records	{ name: string }	// { name: John }
Variants	["Foo"] ["Bar", nat]	// ["Bar", 1 as nat]

JsLIGO :: Type Examples

```
type buy = {  
    profile: bytes,  
    initial_controller: option<address>  
};
```

```
type update = {  
    id: int,  
    new_owner: address  
};
```

```
type action = | ["Buy", buy]  
              | ["Update", update];
```

JsLIGO Overview :: Constants & Variables

Constants

```
let x = (a : int) : int => {  
  const age : int = 25;  
  return age  
};
```

Variables

```
let add = (a: int, b: int): int => {  
  let c = a;  
  c = c + b;  
  return c  
}
```

JsLIGO Overview :: Case Matching

```
type action = | ["Buy", buy] | ["Update", update];
```

```
// ...
```

```
let main = ([act, storage]: [action, storage]):[list<operation>, storage] => {  
  return match (act, {  
    Buy: b => buy([b, storage]),  
    Update: u => update_owner([u, storage]),  
  });  
};
```

JsLIGO Overview :: Iteration

Terminal Recursion (Functional style)

```
let sum_list = ([l, acc]: [list<int>, int]): int => {  
  return match(l, list([  
    ([] : list<int>)          => acc,  
    ([hd, ...tl] : list<int>) => sum_list (tl, hd + acc)  
  ]));  
}
```

JsLIGO Overview :: Iteration

For Loop (Imperative style)

```
let sum_list = (l: list<int>): int => {  
  let total : int = 0;  
  for (const i of l) {  
    total = total + i  
  }  
  return total  
}
```

JsLIGO Overview :: Namespace (Module)

```
namespace EURO {  
  export type t = nat;  
  export let add = ([a, b]: [t, t]): t => a + b;  
  export let zero: t = 0 as nat;  
  export let one: t = 1 as nat  
}
```

JsLIGO Overview :: Namespace (Module)

```
#import "euro.jsligo" "EURO"

type storage = EURO.t;
type return_ = [list<operation>, storage];

let main = ([action, store]: [unit, storage]): return_ =>
  [list([]) as list (operation),
   EURO.add(store, EURO.one)]
```

JsLIGO Overview :: Namespace (Module)

```
#import "euro.jsligo" "EURO"  
  
type storage = EURO.t;  
type return_ = [list<operation>, storage];  
  
let main = ([action, store]: [unit, storage]): return_ =>  
  [list([]) as list (operation),  
   EURO.add(store, EURO.one)]
```

#import and syntaxes interoperability

Example Contract

Example Contract :: Behavior

```
type storage = int;
type parameter =
  ["Increment", int]
| ["Decrement", int]
| ["Reset"];
type result_ = [list<operation>, storage]
```

```
let main = ([action, store]: [parameter, storage]) : result_ => {
  return [
    list([]) as list<operation>,
    match(action, {
      Increment: (n: int) => store + n,
      Decrement: (n: int) => store - n,
      Reset:      ()      => 0
    })
  ];
};
```



Testing a Contract

Testing a Contract :: Framework

Tests are written in LIGO and work directly with LIGO code

Extra types and operations for manipulating Tezos context

Mutation testing primitives

Generation of random values for a type

Ability to catch errors in transactions

Testing a Contract :: Example

```
#import "contract.jsligo" "Contract"

let _test_increment = () : bool => {

}

let test_increment = _test_increment ();
```

Testing a Contract :: Example

```
#import "contract.jsligo" "Contract"

let _test_increment = () : bool => {
  let initial_storage = 42 as int;
  let [address, _, _] = Test.originate(Contract.main, initial_storage, 0 as tez);

}

let test_increment = _test_increment ();
```

Testing a Contract :: Example

```
#import "contract.jsligo" "Contract"

let _test_increment = () : bool => {
  let initial_storage = 42 as int;
  let [address, _, _] = Test.originate(Contract.main, initial_storage, 0 as tez);
  let contract = Test.to_contract(address);

}

let test_increment = _test_increment ();
```

Testing a Contract :: Example

```
#import "contract.jsligo" "Contract"

let _test_increment = () : bool => {
  let initial_storage = 42 as int;
  let [address, _, _] = Test.originate(Contract.main, initial_storage, 0 as tez);
  let contract = Test.to_contract(address);
  let r = Test.transfer_to_contract_exn(contract, (Increment (1)), 1 as mutez);

}

let test_increment = _test_increment ();
```

Testing a Contract :: Example

```
#import "contract.jsligo" "Contract"

let _test_increment = () : bool => {
  let initial_storage = 42 as int;
  let [address, _, _] = Test.originate(Contract.main, initial_storage, 0 as tez);
  let contract = Test.to_contract(address);
  let r = Test.transfer_to_contract_exn(contract, (Increment 1)), 1 as mutez);
  return (Test.get_storage(address) == initial_storage + 1);
}

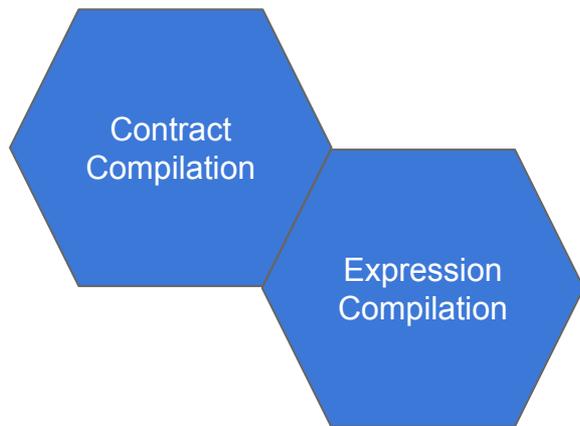
let test_increment = _test_increment();
```

LIGO Tools

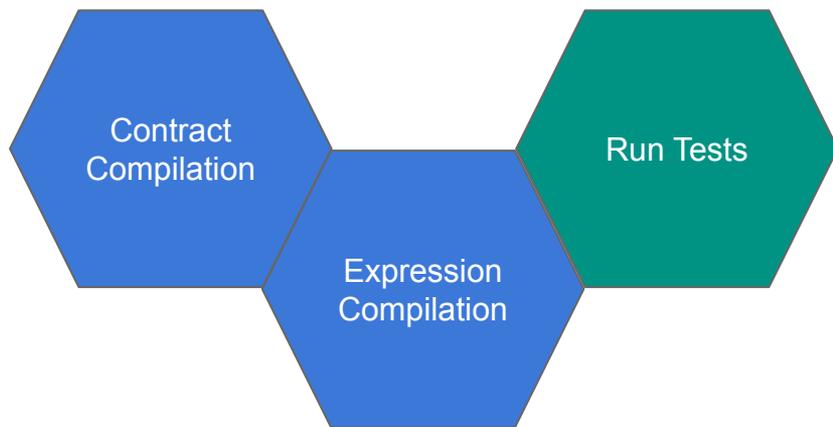
LIGO Tools :: CLI



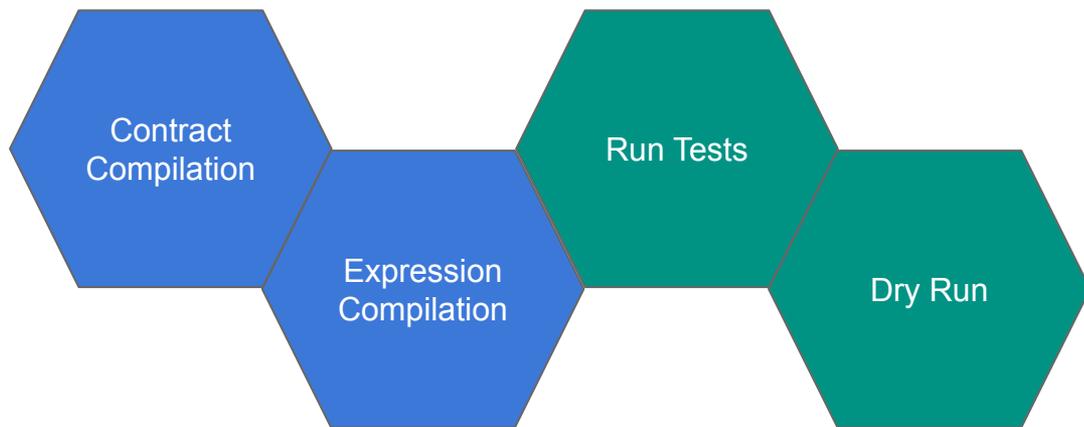
LIGO Tools :: CLI



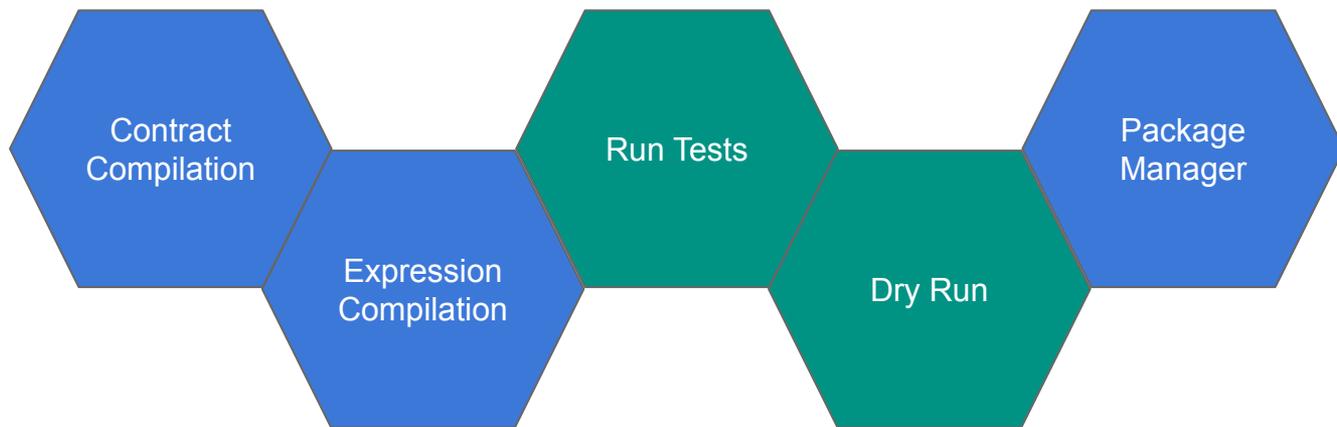
LIGO Tools :: CLI



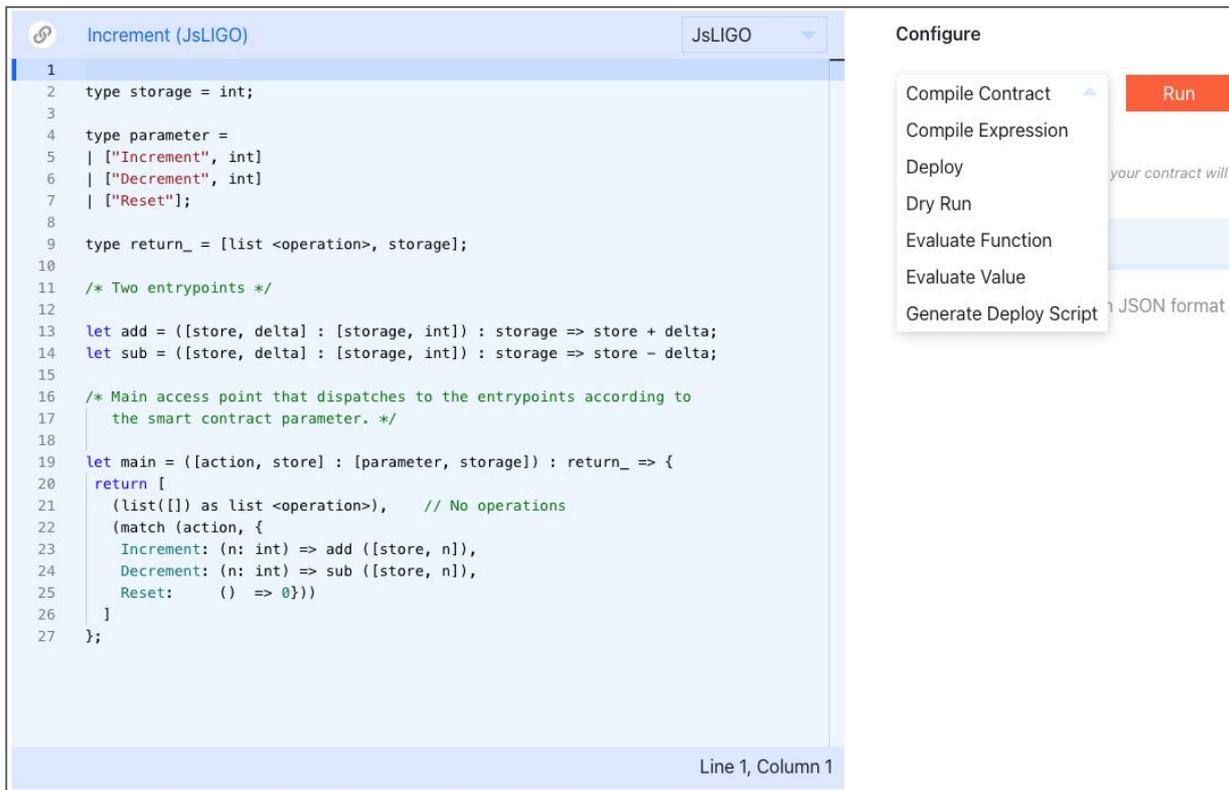
LIGO Tools :: CLI



LIGO Tools :: CLI



LIGO Tools :: Web IDE (ide.ligolang.org)



The screenshot displays the LIGO Tools Web IDE interface. The main editor shows a JsLIGO contract named "Increment" with the following code:

```
1
2 type storage = int;
3
4 type parameter =
5 | ["Increment", int]
6 | ["Decrement", int]
7 | ["Reset"];
8
9 type return_ = [list <operation>, storage];
10
11 /* Two entrypoints */
12
13 let add = ([store, delta] : [storage, int]) : storage => store + delta;
14 let sub = ([store, delta] : [storage, int]) : storage => store - delta;
15
16 /* Main access point that dispatches to the entrypoints according to
17 the smart contract parameter. */
18
19 let main = ([action, store] : [parameter, storage]) : return_ => {
20   return [
21     (list([]) as list <operation>), // No operations
22     (match (action, {
23       Increment: (n: int) => add ([store, n]),
24       Decrement: (n: int) => sub ([store, n]),
25       Reset:    () => 0}))
26   ]
27 };
```

At the bottom right of the editor, it indicates "Line 1, Column 1".

On the right side, there is a "Configure" panel with a dropdown menu open, showing the following options:

- Compile Contract
- Compile Expression
- Deploy
- Dry Run
- Evaluate Function
- Evaluate Value
- Generate Deploy Script

A red "Run" button is visible next to the dropdown menu.

LIGO Tools :: VSCode

```
type storage = int

type parameter =
  | Increment of int
  | Decrement of int
  | Reset

type return = operation list * storage

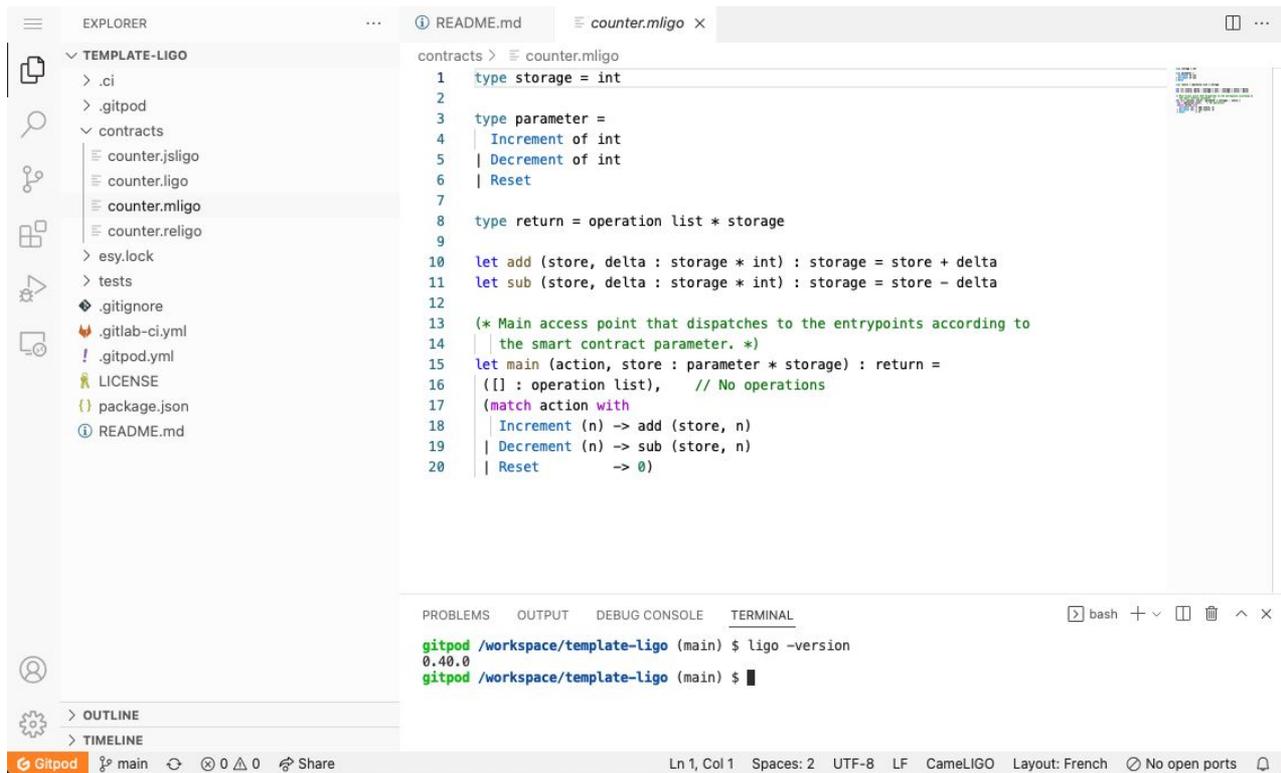
// Two entrypoints

let add (store, delta : storage * int) : storage = store + delta
let sub (store, delta : storage * int) : storage = store - delta

(* Main access point that dispatches to the entrypoints according to
   the smart contract parameter. *)

let main (action, store : parameter * storage) : return =
  ([] : operation list), // No operations
  (match action with
   | Increment (n) -> add (store, n)
   | Decrement (n) -> sub (store, n)
   | Reset -> 0)
```

LIGO Tools :: Gitpod



The screenshot displays a Gitpod workspace environment. On the left, the Explorer sidebar shows a project structure for 'TEMPLATE-LIGO' with folders like '.ci', '.gitpod', 'contracts', 'esy.lock', 'tests', and files like '.gitignore', 'LICENSE', and 'README.md'. The main editor area shows the 'counter.mligo' file with the following code:

```
contracts > counter.mligo
1  type storage = int
2
3  type parameter =
4  | Increment of int
5  | Decrement of int
6  | Reset
7
8  type return = operation list * storage
9
10 let add (store, delta : storage * int) : storage = store + delta
11 let sub (store, delta : storage * int) : storage = store - delta
12
13 (* Main access point that dispatches to the entrypoints according to
14    the smart contract parameter. *)
15 let main (action, store : parameter * storage) : return =
16   ((): operation list), // No operations
17   (match action with
18    | Increment (n) -> add (store, n)
19    | Decrement (n) -> sub (store, n)
20    | Reset         -> 0)
```

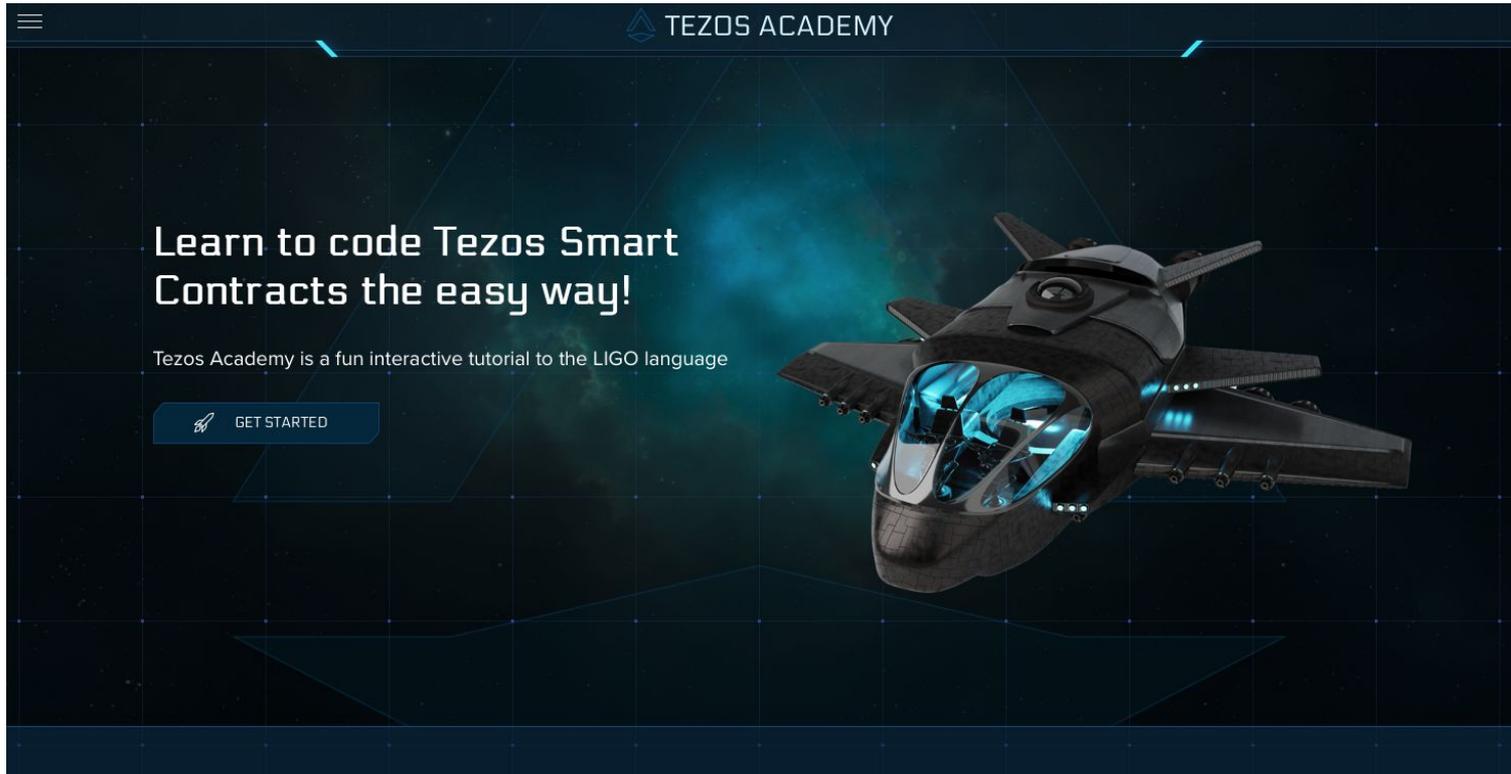
At the bottom, the Terminal panel shows the command `ligo -version` being executed, resulting in the output `0.40.0`.

Gitpod status bar: Ln 1, Col 1 Spaces: 2 UTF-8 LF CameLIGO Layout: French No open ports

LIGO Tools :: And more ...



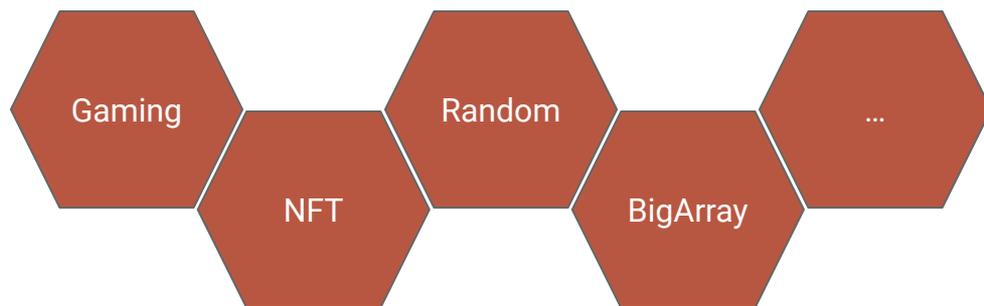
LIGO Tools :: Tutorial (academy.ligolang.org)



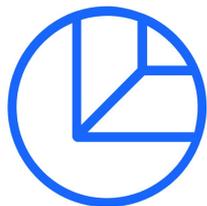
LIGO Catalog

LIGO Catalog :: Contract Library (Soon)

Smart contracts templates + Documentations



ligolib.org



LIGOLANG
A MARIGOLD PROJECT

“A programming language for writing Tezos smart contracts”

ligolang.org