Leo Lang
Syntax Cheat Sheet

Statically Typed	**Pass by Value**	**i8-i128 and u8-u128**
Explicit Types	**“bool” required**	**No type casting**
Groups, Fields & Scalars	**Address’s are 63 bit-hash’s**	`u8=2u8;` *explicit*. `u8=2;` *implicit*.

`src/main.leo`

Functional program

```
// Propose a new proposal to vote on.
transition propose(public info: ProposalInfo) -> Proposal {
    // Authenticate proposer.
    console.assert_eq(self.caller, info.proposer);

    // Generate a new proposal id.
    let id: field = BHP256::hash(info.title);

    // Finalize the proposal id.
    async finalize(id);

    // Return a new record for the proposal.
    return Proposal {
        owner: self.caller,
        gates: 0u64,
        id,
        info,
    };
}
// Create a new proposal in the "tickets" mapping.
finalize propose(public id: field) {
    increment(tickets, id, 0u64);
}
```

`build/main.aleo`

Leo Instructions

```
function propose:
    input r0 as ProposalInfo.public;
    assert.eq self.caller r0.proposer;
    hash.bhp256 r0.title into r1;
    cast self.caller 0u64 r1 r0 into r2 as Proposal.record;
    output r2 as Proposal.record;
    finalize r1;

finalize propose:
    input r0 as field.public;
    increment tickets[r0] by 0u64;
```

Leo vs Aleo instruction comparison

Import = import {filename}.leo;

Program ID = program {name}.{network} **{...}**

Mapping = mapping {name}: {key} => {value};

Record = record {name} **{** {name}: {type}, **}**

Struct = struct {name} **{** {name}: {type}, **}**

Transitions = transition {name} **(** {visibility} {name}: {type}, … **) ->** {return type} **{**
return {expression};
}

Finalize = finalize {name}:

Increment/decrement = increment/decrement (mapping, key, value);


```
import foo.leo;

program hello.aleo {
    mapping balances: address => u64;

    record token {
        owner: address,
        gates: u64,
        amount: u64,
    }

    struct message {
        sender: address,
        object: u64,
    }

    transition mint_public(
        public receiver: address,
        public amount: u64,
    ) -> token {
        async finalize(receiver, amount);
        return token {
            owner: receiver,
            gates: 0u64,
            amount,
        };
    }

    finalize mint_public(
        public receiver: address,
        public amount: u64,
    ) {
        increment(balances, receiver, amount);
    }

    function compute(a: u64, b: u64) -> u64 {
        return a + b;
    }
}
```

Leo Language Layout