New version of this cheat-sheet :

https://gl.githack.com/ligo.suzanne.soy/training-gitpod/raw/master/cheat-sheet-jsligo.html

Basic overview of JsLIGO

This document only lists a subset of the syntax and features of the language, but contains everything that is covered as part of the training and is required to solve the exercises.

Important: in all the examples below, when some text is between square brackets and italics, *[like this]*, all of it should be replaced by the value you need. In particular, you shouldn't type these square brackets.

Basic syntax and types

Structure of a basic Smart-Contract	`const main = ([param, oldStorage] : [int, int]) : [list<operation>, int] => {`
	`  let newStorage = oldStorage + param;`
	`  return [list([]) as list<operation>, newStorage];`
	`}`

Definition of a value	`let` *`[value name]`* `:` *`[name of type]`* `=` *`[value];`*
Array	`[` *`[Value 1]`*`,` *`[Value 2]`*`,` *`[...]`* `]`
Annotated value	`([value] as [type])`

Tuple annotated with its type	`[[value 1], [value 2], [...] ] as [ [type 1], [type 2], [...] ]`
Integer type	Type `int`, Examples: `42, -38` No limit to the range of value, negative like positive.
Natural type	Type `nat`, Example: `7 as nat` Positive number, no upper limit to the value
Tezos tokens type	Type `tez`, Examples: `0.34 as tez, 340000 as mutez` Range limited by 64 bits (may change to no limit in future protocols), integer number of millionths of tez

Arithmetic operations	`let a : int = 5 + 10;`
	`let b : nat = (5 as nat) + (3 as nat);`
	`let c : int = (10 as nat) - (3 as nat);`
	`let d : tez = (5 as tez) - (4 as tez);`
	`let e : int = 10 * 4;`
	`let g : int = 20 / 3;        // gives 6`
	`let h : nat = 20 % 3;      // gives 2`

String type	Type `string`, Examples :`"Hello"` No limit to the length.
String concatenation	`let a : string = "Hello" + " " + "World!";        // Gives "Hello World!"`

Dry-run, simple compilation

Dry-run of a contract	`ligo run dry-run [.jsligo file] -e main '[Value of parameter]' '[Initial value of storage]'`
Dry-run with a tuple	`ligo run dry-run two_numbers.jsligo -e main '[[value 1], [value 2]]' '[[value 3], [value 4]]'`
Dry-run with strings	`ligo run dry-run strings.jsligo -e main '"[text]"' '"[text]"'`
Compilation of a contract	`ligo compile contract [.jsligo file] -e main > [.tz file]`

More advanced syntax

Type alias	type *[name of alias] = [description of type];*
	Example*:* type storage = [int, string];

Simple variant type	type *[name of alias]* = | ["*[First name]*"] | ["*[Second name]*"] | ["*[...]*"] Example: type color = | ["White"] | ["Black"] | ["Red"]
Example use of a variant type	const myColor : color = White();

Simple pattern matching	match(*[matched value]*, {
	      [value 1] : () => *[expression 1]*,
	      [value 2] : () => *[expression 2]*,
	      [...]
	});
	
	Example:
	let newNbWhite = match(param, {
	   Black: () => nbWhite,
	   White: () => nbWhite + 1,
	   Red: () => nbWhite
	});

Variant type with associated value	type [name of alias] = | ["*[First name]*", *[associated type]*] | ["*[Second name]*", *[associated type]*] | *[...]* Exemple : type couleur = | ["White", nat] | ["Black"] | ["Red", nat, int];

<table>
<tr><td>Example use of variant type with value</td><td><pre>const myColor : color = White(10 as nat);</pre></td></tr>
<tr><td>Entry points: example</td><td><pre>type action =
   ["Increment", nat]
 | ["Decrement", nat];

const add = ([oldStor, value]: [int, nat]) : int => oldStor + value;
const sub = ([oldStor, value]: [int, nat]) : int => oldStor - value;

const main = ([parameter, oldStor] : [action, int]) : [list<operation>, int] => {
    let newStor = match(parameter, {
     Increment: (n: nat) => add(oldStor, n),
     Decrement: (n: nat) => sub(oldStor, n)
    });
    return [list([]) as list<operation>, newStor];
}</pre></td></tr>
</table>

Compilation of parameter and storage

Compilation of parameter from LIGO to Michelson	`ligo compile parameter [.jsligo file] --entry-point main '[Value of parameter in LIGO]'`
Compilation of storage from LIGO to Michelson	`ligo compile storage [.jsligo file] --entry-point main '[Value of parameter in LIGO]'`
If it’s a string	`Remember to write '"[Value of the string]"'`

Boolean conditions, verifications

Boolean values, Boolean operators	`let logical_and: bool = true && true;`
	`let logical_or: bool = false || true;`
	`let logical_not: bool = !false;`
	`let gt: bool = 4 > 3;`
	`let lt: bool = 4 < 3;`
	`let gte: bool = 4 >= 3;`
	`let equal: bool = 4 == 4;`
	`Let different: bool = 4 != 5;`

Conditional instruction	`if ([condition]) {`
	`   [instructions if condition is true]`
	`} else {`
	`   [instructions if condition is false]`
	`}`

Errors	`failwith("[message]") as [type];`

Addresses

Hard-coded address	`("tz1KqTpEZ7Yob7QbPE4Hy4Wo8fHG8LhKxZSx" as address)`
Direct caller	`Tezos.get_sender()`

Original caller	`Tezos.get_source()`
Contract itself	`Tezos.get_self_address()`

Timestamp

Hardcoded timestamp	`("2021-05-10t11:00:00Z" as timestamp)`
Current block date and time	`Tezos.get_now()`
Add or subtract seconds	`Tezos.get_now() - 3600`

Options

Definition	`type [name of alias] = option<[type]>;`
No value	`None() as option<[type]>;`
Some value	`Some([value])`

Extracting a value	`match([option value], {`
	`   Some: ([name] : [type]) => result,`
	`   None: () => (failwith "[Message]" as [type])`
	`});`

Transactions

Transfer some tokens	`let op : operation = Tezos.transaction(unit, [amount] as tez, [destination contract]);`
Get the contract from an address	`let owner_contract_opt = Tezos.get_contract_opt (owner) as option<contract<unit>>;`
Type of contract without a parameter	`contract<unit>`

Maps

Define a type of map	`type [alias name] = map<[key type], [value type]>;`
Create an empty map	`let [variable name]: [type of map] = Map.empty;`

Initialize a map with a few values.	`let [variable name] : [type of map] =`
	`  Map.literal (list([`
	`    [ [key 1], [value 1] ],`
	`    [ [key 2], [value 2] ],`
	`    ...`
	`  ]));`

Access to a map entry	`let [variable name]: option<[value type]> = Map.find_opt([key], [map variable]);`

Access to a map entry and extraction of the value from the option.	`match(Map.find_opt ([clé], [variable map]), {`
	`    Some: ([value name]: [value type]) => [value name],`
	`    None: () => (failwith("Not found") as [value type])`
	`  });`

Test if a map contains a given entry.	`if (Map.find_opt([key], [map variable]) == None() as [value type]) {`
	`    ...`
	`}`

Add, update or delete a value for a given key.	`Map.update([key], [option on the value], [the map]);`
	`// Delete if the option is None() as [value type];`

Records

Declare a record type	`type [alias name] = {`
	`   [property name] : [property type],`
	`   [property name] : [property type],`
	`   ...`
	`};`
		Example:
		`type person = {`
		`   firstName : string,`
		`   lastName : string,`
		`   age : int`
		`};`

Create a value with that type	`let [variable name] : [type name] = {`
	`  [property name] : [value],`
	`  [property name] : [value],`
	`  ...`
	`};`
		Example:
		`let alice : person = {`
		`  firstName : "Alice",`
		`  lastName : "Smith",`
		`  age : 28`
		`};`

Read access to a property	`let [variable name] : [type] = [name of record variable].[property name];`
	Example:
	`let aliceLastName : string = alice.lastName;`

Modify the value of one or more properties.	`[record variable name] = ({`
	`...[record variable name],`
	`[property name]: [new value],`
	`[property name]: [new value]`
	`});`
	Example:
	`alice = ({`
	`...alice,`
	`age: alice.age + 1,`
	`lastName: "Durand"`
	`});`

Test Scenarios

General structure of the test.	`#include "./contract.jsligo"`
	
	`let test_code = (): bool => {`
	`   ...`
	`}`
	`let test = test_code();`

Origination of a contract.	`let init_storage = ... ;`
	`let originated_contract = Test.originate(main, init_storage, 0 as tez);`
	`let addr = originated_contract[0];`
	`let contr = Test.to_contract(addr);`

Test of a call to an entry point.	`Test.log(Test.transfer_to_contract(contr, (Increment(32)), 10 as tez));`
Displaying the content of the storage.	`Test.log(Test.get_storage(addr));`

Comparing the storage to an expected value.	`let result = Test.get_storage(addr);`
	`return Test.michelson_equal(result, (32 as int));`

Obtaining an address for a test account.	`let owner = Test.nth_bootstrap_account(0);`
Define the source address of the next call to a contract.	`Test.set_source(owner);`
Define the date that is simulated for the next call to a contract.	`Test.set_now("2021-06-28t11:00:00Z" as timestamp);`
Obtain the balance of a contract	`Test.get_balance`
Run the test from the command line	`ligo run test` *[path of .jsligo file]*