Michelson Language Cheat Sheet

Type taxonomy

int, nat, timestamp ,mutez	numeric
string, bytes	sequences
list, set, map, big_map	structural

Types

address	address of untyped contract
bigmap kty vty	big map
bool	`true` or `false`
bytes	sequence of bytes
chain_id	chain identifier
contract `type`	address of contract w param type `type`
int	arbitrary precision integer
key	public key
key_hash	hash of public key
lambda `ty1` ty2	lambda with given param & return types
list `type`	single immutable homogenous list
map `ktv` vty	immutable map from `kty` to `vty`
mutez	type for manipulating tokens
nat	arbitrary precision natural number
operation	internal operation emitted by contract
option `type`	optional value
or `ty1` ty2	union of two types
pair `ty1` ty2	pair of two values
set `cty`	immutable set of comparable `ctys`
signature	cryptographic signature
string	string of characters
timestamp	real-world date
unit	the type whose only value is `unit`

Instructions

ABS	Obtain the absolute value of an integer
ADD	Add two numerical values
ADDRESS	Push the address of a contract
AMOUNT	Push the amount of the current transaction
AND	Boolean and bitwise AND
APPLY	Partially apply a tuplified function from the stack
BALANCE	Push the current amount of mutez of the executing contract
BLAKE2B	Compute a Blake2B cryptographic hash
CAR	Access the left part of a pair
CDR	Access the right part of a pair
CHAIN_ID	Push the chain identifier
CHECK_SIGNATURE	Verify “signature“ of “bytes“ by “key“
COMPARE	Compare two values
CONCAT	Concatenate a string, byte sequence, string list or byte sequence list
CONS	Prepend an element to a list
CONTRACT	Cast an address to a typed contract
CREATE_CONTRACT	Push a contract creation operation
DIG	Retrieve the “n“ th element of the stack
DIP	Run code protecting the top of the stack
DROP	Drop the top “n“ elements of the stack
DUG	Insert the top element at depth “n“
DUP	Duplicate the top of the stack
EDIV	Euclidean division
EMPTY_BIG_MAP	Build a new, empty “big_map“ from “kty“ to “vty“
EMPTY_MAP	Build a new, empty “map“ from “kty“ to “vty“
EMPTY_SET	Build a new, empty set for elements of type “cty“
EQ	Check that the top of the stack EQuals zero
EXEC	Execute a function from the stack
FAILWITH	Explicitly abort the current program
GE	Check that the top of the stack is Greater Than or Equal to zero
GET	Access an element in a “map“ or “big_map“
GT	Check that the top of the stack is Greater Than zero
HASH_KEY	Compute the Base58Check of a public key
IF	Conditional branching
IF_CONS	Inspect a list
IF_LEFT	Inspect a value of a union
IF_NONE	Inspect an optional value
IMPLICIT_ACCOUNT	Create an implicit account
INT	Convert a natural number to an integer
ISNAT	Convert a non-negative integer to a natural number
ITER	Iterate over a “set“, “list“ or “map“
LAMBDA	Push a lambda onto the stack
LE	Check that the top of the stack is Less Than or Equal to zero
LEFT	Wrap a value in a union (left case)
LOOP	A generic loop
LOOP_LEFT	Loop with accumulator
LSL	Logically left shift a natural number
LSR	Logically right shift a natural number
LT	Check that the top of the stack is Less Than zero
MAP	Apply the body expression to each element of a “list“ or “map“.
MEM	Check for the presence of a binding for a key in a “map“, “set“ or “big_map“
MUL	Multiply two numerical values
NEG	Negate a numerical value
NEQ	Check that the top of the stack does Not EQual zero
NIL	Push an empty list
NONE	Push the absent optional value
NOOP	Empty instruction sequence
NOT	Boolean negation and bitwise complement
NOW	Push block timestamp
OR	Boolean and bitwise OR
PACK	Serialize data
PAIR	Build a pair from the stack’s top two elements
PUSH	Push a constant value of a given type onto the stack
RIGHT	Wrap a value in a union (right case)
SELF	Push the current contract
SENDER	Push the contract that initiated the current internal transaction
SEQ	Instruction sequence
SET_DELEGATE	Push a delegation operation
SHA256	Compute a SHA-256 cryptographic hash
SHA512	Compute a SHA-512 cryptographic hash
SIZE	Obtain size of a “string“, “list“, “set“, “map“ or byte sequence
SLICE	Obtain a substring or subsequence of a “string“ respectively byte sequence “bytes“
SOME	Wrap an existing optional value
SOURCE	Push the contract that initiated the current transaction
SUB	Subtract two numerical values
SWAP	Swap the top two elements of the stack
TRANSFER_TOKENS	Push a transaction operation
UNIT	Push the unit value onto the stack

UNPACK	Deserialize data, if valid
UPDATE	Add or remove an element in a “map“, “big_map“ or “set“
XOR	Boolean and bitwise eXclusive OR

Instructions (detail)

ABS ctx :- ABS :: int : A => nat : A | **ADD** ctx :- ADD :: nat : nat : A => nat : A |ctx :- ADD :: nat : int : A => int : A |ctx :- ADD :: int : nat : A => int : A |ctx :- ADD :: int : int : A => int : A |ctx :- ADD :: timestamp : int : A => timestamp : A |ctx :- ADD :: int : timestamp : A => timestamp : A |ctx :- ADD :: mutez : mutez : A => mutez : A | **ADDRESS** ctx :- ADDRESS :: contract ty1 : A => address : A | **AMOUNT** ctx :- AMOUNT :: A => mutez : A | **AND** ctx :- AND :: bool : bool : A => bool : A |ctx :- AND :: nat : nat : A => nat : A |ctx :- AND :: int : nat : A => nat : A | **APPLY** ctx :- APPLY :: ty1 : lambda (pair ty1 ty2) ty3 : A => lambda ty2 ty3 : A | **BALANCE** ctx :- BALANCE :: A => mutez : A | **BLAKE2B** ctx :- BLAKE2B :: bytes : A => bytes : A | **CAR** ctx :- CAR :: pair ty1 ty2 : A => ty1 : A | **CDR** ctx :- CDR :: pair ty1 ty2 : A => ty2 : A | **CHAIN_ID** ctx :- CHAIN_ID :: A => chain_id : A | **CHECK_SIGNATURE** ctx :- CHECK_SIGNATURE :: key : signature : bytes : A => bool : A | **COMPARE** ctx :- COMPARE :: cty : cty : A => int : A | **CONCAT** ctx :- CONCAT :: string : string : A => string : A |ctx :- CONCAT :: list string : A => string : A |ctx :- CONCAT :: bytes : bytes : A => bytes : A |ctx :- CONCAT :: list bytes : A => bytes : A | **CONS** ctx :- CONS :: ty1 : list ty1 : A => list ty1 : A | **CONTRACT** ctx :- CONTRACT ty1 :: address : A => option (contract ty1) : A | **CREATE_CONTRACT** ctx :- CREATE_CONTRACT ty1 ty2 code :: option key_hash : mutez : ty1 : A => operation : address : A | **DIG** ctx :- DIG n :: A @ (ty1 : B) => ty1 : (A @ B) | **DIP** ctx :- DIP n code :: A @ B => A @ C | **DROP** ctx :- DROP n :: A @ B => B | **DUG** ctx :- DUG n :: ty1 : (A @ B) => A @ (ty1 : B) | **DUP** ctx :- DUP :: ty1 : A => ty1 : ty1 : A | **EDIV** ctx :- EDIV :: nat : nat : A => option (pair nat nat) : A |ctx :- EDIV :: nat : int : A => option (pair int nat) : A |ctx :- EDIV :: int : nat : A => option (pair int nat) : A |ctx :- EDIV :: int : int : A => option (pair int nat) : A |ctx :- EDIV :: mutez : nat : A => option (pair mutez mutez) : A |ctx :- EDIV :: mutez : mutez : A => option (pair nat mutez) : A | **EMPTY_BIG_MAP** ctx :- EMPTY_BIG_MAP kty vty :: A => big_map kty vty : A | **EMPTY_MAP** ctx :- EMPTY_MAP kty vty :: A => map kty vty : A | **EMPTY_SET** ctx :- EMPTY_SET cty :: A => set cty : A | **EQ** ctx :- EQ :: int : A => bool : A | **EXEC** ctx :- EXEC :: ty1 : lambda ty1 ty2 : A => ty2 : A | **FAILWITH** ctx :- FAILWITH :: ty1 : A => B | **GE** ctx :- GE :: int : A => bool : A | **GET** ctx :- GET :: kty : map kty vty : A => option vty : A |ctx :- GET :: kty : big_map kty vty : A => option vty : A | **GT** ctx :- GT :: int : A => bool : A | **HASH_KEY** ctx :- HASH_KEY :: key : A => key_hash : A | **IF** ctx :- IF code1 code2 :: bool : A => B | **IF_CONS** ctx :- IF_CONS code1 code2 :: list ty1 : A => B | **IF_LEFT** ctx :- IF_LEFT code1 code2 :: or ty1 ty2 : A => B | **IF_NONE** ctx :- IF_NONE code1 code2 :: option ty1 : A => B | **IMPLICIT_ACCOUNT** ctx :- IMPLICIT_ACCOUNT :: key_hash : A => contract unit : A | **INT** ctx :- INT :: nat : A => int : A | **ISNAT** ctx :- ISNAT :: int : A => option nat : A | **ITER** ctx :- ITER code :: list ty1 : A => A |ctx :- ITER code :: set cty : A => A |ctx :- ITER code :: map kty vty : A => A | **LAMBDA** ctx :- LAMBDA ty1 ty2 code :: A => lambda ty1 ty2 : A | **LE** ctx :- LE :: int : A => bool : A | **LEFT** ctx :- LEFT ty2 :: ty1 : A => or ty1 ty2 : A | **LOOP** ctx :- LOOP code :: bool : A => A | **LOOP_LEFT** ctx :- LOOP_LEFT code :: or ty1 ty2 : A => ty2 : A | **LSL** ctx :- LSL :: nat : nat : A => nat : A | **LSR** ctx :- LSR :: nat : nat : A => nat : A | **LT** ctx :- LT :: int : A => bool : A | **MAP** ctx :- MAP code :: list ty1 : A => list ty2 : A |ctx :- MAP code :: map kty ty1 : A => map kty ty2 : A | **MEM** ctx :- MEM :: cty : set cty : A => bool : A |ctx :- MEM :: kty : map kty vty : A => bool : A |ctx :- MEM :: kty : big_map kty vty : A => bool : A | **MUL** ctx :- MUL :: nat : nat : A => nat : A |ctx :- MUL :: nat : int : A => int : A |ctx :- MUL :: int : nat : A => int : A |ctx :- MUL :: int : int : A => int : A |ctx :- MUL :: mutez : nat : A => mutez : A |ctx :- MUL :: nat : mutez : A => mutez : A | **NEG** ctx :- NEG :: nat : A => int : A |ctx :- NEG :: int : A => int : A | **NEQ** ctx :- NEQ :: int : A => bool : A | **NIL** ctx :- NIL ty1 :: A => list ty1 : A | **NONE** ctx :- NONE ty1 :: A => option ty1 : A | **NOOP** ctx :- :: A => A | **NOT** ctx :- NOT :: bool : A => bool : A |ctx :- NOT :: nat : A => int : A |ctx :- NOT :: int : A => int : A | **NOW** ctx :- NOW :: A => timestamp : A | **OR** ctx :- OR :: bool : bool : A => bool : A |ctx :- OR :: nat : nat : A => nat : A | **PACK** ctx :- PACK :: ty1 : A => bytes : A | **PAIR** ctx :- PAIR :: ty1 : ty2 : A => pair ty1 ty2 : A | **PUSH** ctx :- PUSH ty1 x :: A => ty1 : A | **RIGHT** ctx :- RIGHT ty1 :: ty2 : A => or ty1 ty2 : A | **SELF** ctx :- SELF :: A => contract ty : A | **SENDER** ctx :- SENDER :: A => address : A | **SEQ** ctx :- code1 ; code2 :: A => C | **SET_DELEGATE** ctx :- SET_DELEGATE :: option key_hash : A => operation : A | **SHA256** ctx :- SHA256 :: bytes : A => bytes : A | **SHA512** ctx :- SHA512 :: bytes : A => bytes : A | **SIZE** ctx :- SIZE :: set cty : A => nat : A |ctx :- SIZE :: map kty vty : A => nat : A |ctx :- SIZE :: list ty1 : A => nat : A |ctx :- SIZE :: string : A => nat : A |ctx :- SIZE :: bytes : A => nat : A | **SLICE** ctx :- SLICE :: nat : nat : string : A => option string : A |ctx :- SLICE :: nat : nat : bytes : A => option bytes : A | **SOME** ctx :- SOME :: ty1 : A => option ty1 : A | **SOURCE** ctx :- SOURCE :: A => address : A | **SUB** ctx :- SUB :: nat : nat : A => int : A |ctx :- SUB :: nat : int : A => int : A |ctx :- SUB :: int : nat : A => int : A |ctx :- SUB :: int : int : A => int : A |ctx :- SUB :: timestamp : int : A => timestamp : A |ctx :- SUB :: timestamp : timestamp : A => int : A |ctx :- SUB :: mutez : mutez : A => mutez : A | **SWAP** ctx :- SWAP :: ty1 : ty2 : A => ty2 : ty1 : A | **TRANSFER_TOKENS** ctx :- TRANSFER_TOKENS :: ty1 : mutez : contract ty1 : A => operation : A | **UNIT** ctx :- UNIT :: A => unit : A | **UNPACK** ctx :- UNPACK ty1 :: bytes : A => option ty1 : A | **UPDATE** ctx :- UPDATE :: cty : bool : set cty : A => set cty : A |ctx :- UPDATE :: kty : option vty : map kty vty : A => map kty vty : A |ctx :- UPDATE :: kty : option vty : big_map kty vty : A => big_map kty vty : A | **XOR** ctx :- XOR :: bool : bool : A => bool : A |ctx :- XOR :: nat : nat : A => nat : A |