# Literals and Identifiers

## Number literals

FunC allows decimal and hexadecimal integer literals (leading zeros are allowed).

For example, `0` , `123` , `-17` , `00987` , `0xef` , `0xEF` , `0x0` , `-0xfFAb` , `0x0001` , `-0` , `-0x0` are valid number literals.

## String literals

Strings in FunC are quoted in double quotes `"` like `"this is a string"` . Special symbols like `\n` and multi-line stings are not supported. String are used only in asm functions definitions.

## Identifiers

FunC allows a really wide class of identifiers (functions and variables names). Namely, any (single-line) string which doesn't contain special symbols `;` , `,` ,

`(` , `)` , (space or tab), `~` and `.` , doesn't start as comment or string literal (with `"` ), isn't a number literal, isn't an underscore `_` and isn't a keyword is a valid identifier (with the only exception that if it starts with `` ` `` , it must end with the same `` ` `` and can't contain any other `` ` `` except for this two).

Also function names in function definitions may start with `.` or `~` .

For example, those are valid identifier:

- `query` , `query'` , `query''`
- `elem0` , `elem1` , `elem2`
- `CHECK`
- `_internal_value`
- `message_found?`
- `get_pubkeys&signatures`
- `dict::udict_set_builder`
- `_+_` (the standard addition operator of type `(int, int) -> int` in prefix notation, although it is already defined)
- `fatal!`

`'` at the end of the name of a variable is conventionally used when some modified version of the old value is introduced. For example, almost all modifying built-in primitives for hashmap manipulation (except ones with prefix

`~` ) take a hashmap and return a new version of the hashmap along with some other data, if necessary. It is convenient to name those values with the same name suffixed by `'` .

Suffix `?` is usually used for boolean variables (TVM hasn't built-in type bool; bools are represented by integers: 0 is false and -1 is true) or for functions that returns some flag, usually indicating success of the operation (like `udict_get?` from stdlib.fc).

Those are not valid identifiers:

- `take(first)Entry`
- `"not_a_string`
- `msg.sender`
- `send_message,then_terminate`
- `_`

Some more weird examples of valid identifiers:

- `123validname`
- `2+2=2*2`
- `-alsovalidname`
- `0xefefefhahaha`
- `{hehehe}`

- `pa{--}in"`aaa`"`

Those also are not valid identifier:

- `pa;;in"`aaa`"` (because `;` is prohibited)
- `{-aaa-}`
- `aa(bb`
- `123` (it's a number)

Also FunC has special type of identifiers, which quoted in back quotes `` ` ``. In the quotes any symbols are allowed except for `\n` and the quotes themself.

For example, `` `I'm a variable too` `` is a valid identifier, as well as `` `any symbols ; ~ () are allowed here...` ``