Warning: for the new syntax, check: https://docs.google.com/document/d/14PTzLLA2blhI39FbmZ99068CG-dKQrupMsyaArGQ9SI/edit?usp=sharing

# SmartPy Cheat sheet

This document doesn't contain everything about the syntax of the language, but should contain everything that is covered during this training, and needed 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.

## Links

| | |
|---|---|
| SmartPy IDE | https://smartpy.io/ide |

## Constructor, storage, entry points

| | |
|---|---|
| Constructor, storage initialization | `import smartpy as sp`<br><br>`class [Name of the class](sp.Contract):`<br>`    def __init__(self, [parameter 1], [...]):`<br>`        self.init([name of field] = [value])` |
| Acces so the storage | `self.data.[name of the field]` |
| Entry point | `@sp.entry_point`<br>`def [name of the entry point](self, [parameter 1], [...]):`<br>`    [code]` |

**Note**: There are two underscores on each side of the word "init": `__init__`, not `_init_`

## Basic types

| | | |
|---|---|---|
| Integer | `sp.TInt` | `-34, -12, sp.int(42)` |
| Natural | `sp.TNat` | `sp.nat(12)` |
| Integer or Natural | `sp.TIntOrNat` | `42` |
| Tez Token | `sp.TTez` | `sp.tez(12), sp.mutez(12000000)` |
| Boolean | `sp.TBool` | `True, False` |
| String | `sp.TString` | `"Hello World!"` |
| Address | `sp.TAddress` | `sp.address("tz1YtuZ4vhzzn7ssCt93Put8U9UJDdvCXci4")` |

## SmartPy Variables

| | |
|---|---|
| Variable creation | *[python variable]* = sp.local('*[name of SmartPy var]*', [*initial value]*) |
| Access to the value of the variable | *[SmartPy variable name]*.value = *[value]* |

## Helping the type inference

| | |
|---|---|
| Explicit type | `sp.int(4), sp.nat(4)` |
| Annotating the type | *sp.set_type([expression], [type])* |
| Examples of annotations | *@sp.entry_point*<br>*def my_entry_point(self, x, y, z):*<br>    *sp.set_type(x, sp.TInt)*<br>    *sp.set_type(y, sp.TString)* |

## Arithmetic operators

```
i = sp.int(5)
```

```
n = sp.nat(3)
a = n + 2  # a has type sp.TNat
b = n - 2  # b has type sp.TInt
c = i * n  # c has type sp.TInt
d = i / n  # integer division (unlike python)
e = i // n # integer division
f = sp.ediv(i, n) # returns a pair that contains the result
                  # of the integer division, and the remainder
g *= 2     # equivalent to g = g * 2
h -= 1     # equivalent to h = h - 1
i = "Hello" + "World" # concatenates two strings

# computation of a ratio between tez
sp.split_tokens(amount, quantity, totalQuantity)
# Computes amount * quantity / totalQuantity where amount is of type
# sp.TMutez, and quantity and totalQuantity are of type sp.TNat.
# for example, sp.split_tokens(amount, 30, 100)
# computes 30% of amount
```

## Test scenarios

| | |
|---|---|
| Definition of a test | `@sp.add_test(name = "`*`[name of the test]`*`")`<br>`def test():` |
| Contract instantiation | `c1 = StoreValue(`*`[initial value of the storage]`*`)` |
| Scenario creation | `scenario = sp.test_scenario()` |
| Adding some html | `scenario.h1("`*`[some text]`*`")`<br>`scenario.h2("`*`[some text]`*`")`<br>`scenario.p("`*`[some text]`*`")` |
| Adding the contract to the scenario | `scenario += c1` |
| Adding a call to an entry point with no parameter | `c1.`*`[entry point]`*`()` |
| Adding a call to an entry point with one parameter | `c1.`*`[entry point]`*`(`*`[value]`*`)` |
| Call to an entry point with several parameters | `c1.`*`[entry point]([param 1 name] = [value], [param 2 name] = [value], ...)`* |
| Verification about the storage content | `scenario.verify(c1.data.`*`[field name]`*` == `*`[value]`*`)` |

## Timestamps

| | |
|---|---|
| Seconds since 01/01/1970 | `sp.timestamp([number of seconds])` |
| List of parameters | `sp.timestamp_from_utc(year, month, day, hours, minutes, seconds)` |
| Date and time of the current block | `sp.now` |
| Adding some time | `d = a.add_seconds(42)`<br>`e = b.add_minutes(15)`<br>`f = c.add_hours(24)`<br>`g = a.add_days(365)` |
| Difference, in seconds | `h = sp.now - g` |

## Pairs

| | |
|---|---|
| Creation of a pair | `p = sp.pair([value 1], [value 2])` |
| First element | `sp.fst(p)` |
| Second element | `sp.snd(p)` |
| Extracting the two values into two python variables | `x1, x2 = sp.match_pair(p)` |

## Options

| | |
|---|---|
| Creation of an option with no value | `o = sp.none` |
| Creation of an option with a value | `o = sp.some([value])` |
| Extract the value of an option<br>Triggers an error if there is none | `v = o.open_some()` |
| Test if an option has a value | `sp.if (o != sp.none):` |

## Addresses, transactions

| | |
|---|---|
| Transfer of tokens | sp.send(*[address], [value in tez]*) |
| Address of the direct caller of the contract | sp.sender |
| Address of the indirect initial caller of the chain of contracts | sp.source |
| Address of the contrat | sp.self_address |
| Amount transferred to the contract | sp.amount |
| Current balance of the contract | sp.balance |

## Verifications, booleans, errors

| | |
|---|---|
| Error | failwith("*[message]*") |
| Verification, without a message | sp.verify(*[condition]*) |
| Verification, with a message | sp.verify(*[condition], [message]*) |
| Boolean operators | a = sp.bool(True)<br>b = ~a        # Not<br>c = a \| b     # Logical or<br>d = a & b     # Logical and<br>e = a ^ c     # Exclusive or |
| Comparisons | <, >, <=, >=, ==, != |
| Conditional instructions | sp.if *[condition]*:<br>   *[code to run if true]*<br>sp.else:<br>   *[code to run if false]* |
| **Warning**<br>Too many parenthesis | sp.if (a == b):<br># This won't work. At time of writing, Smartpy doesn't accept parenthesis around the whole condition |
| **Warning**<br>Combining boolean operators and comparisons | a < b \| b < c # Causes an error<br>(a < b) \| (b < c) # Works<br># This is due to the mix between smartpy code and python code |

## Maps

| | |
|---|---|
| Empty map | `{}, sp.map({})` |
| Pre-filled map | `varMap = sp.map({`<br>   *[key 1]*`:` *[value 1]*`,`<br>   *[key 2]*`:` *[value 2]*`,`<br>   *[...]*<br>`})` |
| Reading an entry | `v = varMap[` *[key]* `]` |
| Adding or updating an entry | `varMap[` *[key]* `] =` *[value]* |
| Testing if an entry exists for a given key | `varMap.contains(`*[key]*`)` |
| Removal of the entry for a key | `del varMap[` *[key]* `]` |

## Records

| | |
|---|---|
| Creation of a record | `varRecord = sp.record(`<br>   *[field 1] = [value 1]*`,`<br>   *[field 2] = [value 2]*`,`<br>   *[...]*<br>`)` |
| Access to a field (read/write) | `varRecord.[field 1] = [value]` |
| Modification of several fields | `sp.modify_record(varRecord,`<br>   *[field 1] = [value 1]*`,`<br>   *[field 2] = [value 2]*<br>`)` |

## Advanced tests

| | |
|---|---|
| Creation of a test account | `account1 = sp.test_account("[name of the account]")` |
| Getting the address of an account | `address1 = account1.address` |
| In one line | `address1 = sp.test_account("[name of account]").address` |
| Context of a call to an entry point | `c1.entrypoint1().run(sender =` *[address]*`,` |

| | |
|---|---|
| | `amount = `*`[value in tez]`*`,`<br>`now = `*`[a timestamp]`*`,`<br>`valid = False)` |
| Specify who calls the entry point | `sender = `*`[address]`* |
| Specify what amount is transferred | `amount = `*`[value in tez]`* |
| Specify what date is simulated during the call | `now = `*`[a timestamp]`*`,` |
| Specify that the test should fail | `valid = False` |
| Check the balance of the contract | `scenario.verify(c1.balance == sp.tez(`*`[value]`*`))` |

## Serialization, Hashing

| | |
|---|---|
| Serialization of a value, returns TBytes | `sp.pack(`*`[value]`*`)` |
| Deserialization of a value | `sp.unpack(`*`[value in TBytes]`*`)` |
| Hashing of a value of type TBytes | `hashedValue = sp.blake2b([value in TBytes])` |
| Hashing of a value that is not of type TBytes | `hashedValue = sp.blake2b(sp.pack([value])` |