# CITYOPT Planning Tool Expression Language

## Introduction

The CITYOPT Planning tool lets users enter the evaluation metrics, constraints and objective functions as mathematical expressions. This document covers the data types and functions available in the expressions.

Technically the expression language is the programming language Python (see <http://www.python.org/>), version 2.7. It is not necessary to understand Python in order to enter mathematical expressions in the CITYOPT Planning tool.

## Expression usage

Metrics are summary values on the basis of which scenarios are analyzed and visualized. For example, the total operational costs in €, or the total reduction in CO2 emissions in kg could be two metrics of interest. A metric expression defines the value of an evaluation metric in terms of model input parameters, output variables, and external parameters.

In addition to metrics, constraints and objective functions are used in the database optimization and scenario generation (genetic algorithm) features of the CITYOPT Planning tool. A constraint expression bounds the allowed values of model input parameters before simulation, or the acceptable values of output variables and metrics after simulation. Instead of entering explicit inequalities or equations, users enter a constraint expression and then a numerical lower bound and upper bound defining the valid range of expression values.

An objective function expression defines an objective function for optimization, in terms of input parameters, output variables, and metrics. In scenario generation by genetic algorithm, multiple objective functions can be used.

In some cases it is necessary that the scenario generation modifies multiple input parameters in a dependent manner, e.g. one input parameter should always be double the value of another. For this purpose, in scenario generation an input parameter can be defined via an *input expression* that depends on decision variables and external parameters.

## Named parameters and variables

The following data can be referenced by name in expressions:

External parameters are time series data such as cost coefficients or CO2 emission parameters that can be used in constraint, metric and objective function expressions, but not inside a simulation model.

Decision variables are the values decided by the genetic algorithm (or other optimization algorithm) in scenario generation. They can be used in constraint and input expressions only. A decision variable can be defined with a name of its own, or it can share a name with an input parameter; in the latter case the decision variable is referenced via a component name as described below.

Input parameters are input values for a simulation model. Each input parameter is associated with a specific model component and must be referenced in the form C.I where C is the component name and I is the input parameter name. Input parameters can be used in constraint, metric and objective function expressions.

Output variables are time series data exported from a simulation model. Each output variable is associated with a model component and must be referenced in the form C.O where C is the component name and O is the output variable name. Output variables can be used in constraint, metric and objective function expressions.

Metrics are available in constraint and objective function expressions.

A summary of which named parameters and variables can be used in which expressions is shown in the following table. Y indicates that a parameter or variable can be used in an expression.

Table The named parameters and variables that are available in each kind of expression.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Named parameter or variable** | **Expression** | | | |
| Input | Constraint | Metric | Objective |
| External parameter | Y | Y | Y | Y |
| Decision variable | Y | Y | - | - |
| Input parameter | - | Y | Y | Y |
| Output variable | - | Y | Y | Y |
| Metric | - | - | - | Y |

## Data types

The following data types can be used for named parameters and variables.

**Double** – Most numerical values in the system are decimal numbers represented as IEEE double precision floating point values. They have approximately 15 digits of precision and can range in magnitude from 10-308 to 10308. Corresponds to the Python type float.

**Integer** – Integers are stored as signed 32-bit machine integers, with a valid range from  
 –2 147 483 648 to 2 147 483 647. As Python supports arbitrary precision integers, integers may exceed the valid range within expressions, and yet cause a failure when attempting to store the result. Corresponds to the Python type int.

**Timestamp** – Moment of time (date and time), stored as a double precision floating point number indicating the number of seconds from the time origin of the simulation model. For convenience, input values can be represented as human-readable strings in the ISO-8601 standard format YYYY-MM-DDThh:mm:ss, e.g. 2015-12-24T23:59:59. Corresponds to a simulation time value of Python type double.

**String** – Text string parameters can be passed to simulation model components and used in expressions. Corresponds to the Python type str.

**TimeSeries/step** – A time series stored as a sequence of points (time, value), considered as a piecewise constant function from time to value. Between two defined points, the value is defined by the point with a smaller time coordinate. Outside the range of defined points the value is 0. Both times and values are of type Double. Represented by a CITYOPT specific Python class TimeSeries.

**TimeSeries/linear** – A time series stored as a sequence of points (time, value), considered as a piecewise linear function from time to value. Between two defined points, the value is defined by linear interpolation. Outside the range of defined points the value is 0. Both times and values are of type Double. Represented by a CITYOPT specific Python class TimeSeries.

**List of Double** – List of decimal numbers, each of type Double as described above. Represented as a sequence of comma-separated values in brackets, e.g. [1.0, 2.0, 3.0]. Corresponds to the Python type list.

**List of Integer** – List of integers, each of type Integer as described above. Represented as a sequence of comma-separated values in brackets, e.g. [1, 2, 3]. Corresponds to the Python type list.

**List of Timestamp** – List of timestamps, each of type Timestamp as described above. Represented as a sequence of comma-separated *strings* in brackets, e.g. [“2015-12-24T23:59:59”, “2016-01-01T12:00:00”]. Corresponds to the Python type list.

## Valid names

User-defined names may contain only underscore characters, letters from A to Z in both lower and upper case, and numbers. A name cannot begin with a number. All other characters are disallowed. For example, the following names are valid: boiler\_temp, pipeLength, f3.

The Python keywords listed in Table 1 are reserved, and cannot be used as the names of components, external parameters, input parameters, output variables, or metrics. The same applies to the global identifiers listed in Table 2, although they can be used as names of input parameters and output variables.

## Arithmetic operators

The following arithmetic operators are available in expressions: + - \* \*\* / // %

The operator \*\* is exponentiation, e.g. 2\*\*3 evaluates to 8.

The operator // is floor division: a//b evaluates to floor(a/b), such that e.g. 11//4 returns 2. Note that the operator / is floating point division even for integer arguments: 11/4 is 2.75.

The operator % is modulo: a%b evaluates to a-(a//b)\*b, such that e.g. 11%4 returns 3.

The above arithmetic operators are also defined for time series, and are applied to the values of the time series. For example, if ts is of type TimeSeries, then ts+2 results in a time series defined at the same time points with values 2 larger than the values of ts.

When both arguments of an arithmetic operator are time series, the operator is evaluated at time coordinates where either argument is defined, interpolating if necessary. Considering time series as continuous functions, the result is exact for addition and subtraction, and only approximate for other operators such as multiplication. [[1]](#footnote-2)

## Time series access

When working with TimeSeries objects, times can be represented either as simulation time, defined as the number of seconds since the simulation model specific *time origin*, or as datetime objects as defined in the Python standard library [2]. The global function todatetime constructs a datetime object from a simulation time value, and the global function tosimtime converts a datetime into a simulation time value.

The domain of a time series refers to the time interval from the first defined point to the last defined point.

In the following table ts is a TimeSeries object. You can substitute ts with any expression that returns a time series: For example, the product of two time series a and b can be integrated with an expression such as integrate(a \* b, 0, Infinity).

Table Expressions that can be used to work with time series.

| **Expression** | **Description** |
| --- | --- |
| integrate(ts, t0, t1)  integrate(ts,  t0, t1, scale) | Integral from t0 to t1 with the time unit scale (default 1). Essentially, the integral is computed with a time unit of 1 second, and the result is divided by scale. The arguments t0 and t1 can be simulation time values or datetime objects. |
| min(ts) | Minimum value of the time series over its domain: in other words, the value 0 outside the domain is ignored.[[2]](#footnote-3) |
| max(ts) | Maximum value of the time series over its domain. |
| mean(ts) | Mean value of the time series over its domain, considering the time series as a continuous function. |
| var(ts) | Variance of the time series over its domain, considering the time series as a continuous function. |
| stdev(ts) | Standard deviation of the time series over its domain, considering the time series as a continuous function. |
| ts.slice(t0, t1) | Returns a time series that is equal to ts from time t0 up to time t1, and 0 elsewhere. The arguments t0 and t1 can be simulation time values or datetime objects. |
| ts.at(times) | Computes the time series values at the specified times, using interpolation if necessary. The argument times can be a sequence of simulation time values or datetime objects. The method returns an array of doubles. |
| ts.times | Array of doubles specifying the defined time points in simulation time. |
| ts.datetimes | An alternative representation for ts.times: List of datetime objects specifying the defined time points. |
| ts.values | Array of doubles containing the time series values at the defined points. |
| ts.iter() | Iterator over (time, value) pairs. Can be useful for non-trivial time series manipulation especially in conjunction with the time series constructors TimeSeries.step and TimeSeries.linear. |
| TimeSeries(degree, times, values) | Constructs a new TimeSeries instance from given data.  degree is 0 for piecewise constant, 1 for piecewise linear interpolation  times is either a sequence of simulation time values or a sequence of datetime objects  values is a sequence of floating point numbers. |
| TimeSeries.step(  iterable) | Constructs a new TimeSeries instance that uses piecewise constant interpolation. The argument is either an iterable of (time, value) pairs or an iterable of (datetime, value) pairs.  Example:  TimeSeries.step((t+1, 2\*v) for (t,v) in ts.iter()) |
| TimeSeries.linear(  iterable) | Constructs a new TimeSeries instance that uses piecewise linear interpolation. The argument is either an iterable of (time, value) pairs or an iterable of (datetime, value) pairs. |

## Functions

The global functions, constants and modules that can be safely used in expressions are listed in Table 3. In addition to the standard Python types int, float, bool, str, list, set, dict and function, in Table 3 any refers to any type, number is any numerical type, and iterable is any iterable type such as list or set. Function argument types are shown in the Identifier column and result types in the Type column. For full details, see the Python standard library documentation for built-in functions and the indicated Python modules [2].

Table 3 Global functions, constants and modules.

| **Identifier** | **Type** | **Description** | **Module** |
| --- | --- | --- | --- |
| abs(int) | int | Absolute value |  |
| abs(float) | float | Absolute value |  |
| acos(float) | float | Arc cosine in radians.  Fails for arguments outside [-1, 1]. | math |
| all(iterable) | bool | Whether all elements of the iterable are true |  |
| any(iterable) | bool | Whether any element of the iterable is true |  |
| asin(float) | float | Arc sine in radians. Fails for arguments outside [-1, 1]. | math |
| atan(float) | float | Arc tangent in radians | math |
| atan2(float, float) | float | Arc tangent in radians for given (y, x) coordinates | math |
| bool(any) | bool | Boolean constructor: True for nonzero argument |  |
| ceil(float) | float | Ceiling function | math |
| cityopt | module | Functions of project CITYOPT | cityopt |
| cmath | module | Mathematical functions for complex numbers | cmath |
| cos(float) | float | Cosine | math |
| cosh(float) | float | Hyperbolic cosine | math |
| datetime(int, int, int, …) | datetime | datetime constructor from given year, month, day, hour, minutes, seconds. Omitted hours, minutes or seconds are interpreted as 0.  Example: datetime(2014,3,31, 23,59,59) | datetime |
| DAY\_S | int | Day in seconds: equal to 86400. | cityopt |
| dict(…) | dict | Dictionary constructor |  |
| enumerate(  iterable) | iterable | Pairs indices 0, 1, … with iterable elements.  Returns a sequence of tuples (index, element). |  |
| exp(float) | float | Exponential function, equivalent to e\*\*x | math |
| False | bool | Boolean constant |  |
| float(any) | float | Float constructor from numerical or string argument. |  |
| floor(float) | float | Floor function | math |
| HOUR\_S | int | Hour in seconds: equal to 3600. | cityopt |
| hypot(float, float) | float | Hypotenuse of right triangle, equals sqrt(x\*\*2 + y\*\*2) | math |
| Infinity | float | Floating point infinity. Equal to float(‘inf’) | cityopt |
| int(any) | int | Integer constructor from numerical or string argument |  |
| integrate(  TimeSeries,  float, float,  float) | float | Integration of time series over an interval | cityopt |
| itertools | module | Functions for building and combining iterators | itertools |
| len(iterable) | int | Length of a sequence |  |
| list(iterable) | list | List constructor |  |
| log(float) | float | Natural logarithm | math |
| log(float, float) | float | Logarithm in the base given by the second argument | math |
| map(function, iterable) | list | Mapping of a sequence through a function |  |
| math | module | Mathematical functions for real numbers | math |
| max(TimeSeries) | float | Maximum of a time series over its domain | cityopt |
| max(iterable) | number | Maximum element of a sequence |  |
| max(number, …) | number | Maximum of the given arguments |  |
| mean(TimeSeries) | float | Mean value of time series over its domain | cityopt |
| mean(iterable) | float | Mean value of a sequence | cityopt |
| min(TimeSeries) | float | Minimum value of a time series over its domain | cityopt |
| min(iterable) | number | Minimum element of a sequence |  |
| min(number, …) | number | Minimum of the given arguments |  |
| MINUTE\_S | int | Minute in seconds: equal to 60. | cityopt |
| NaN | float | Floating point Not-a-Number. Equal to float(‘nan’) |  |
| None | NoneType | Special null value |  |
| pow(number, number) | number | Power function. Equivalent to x\*\*y |  |
| range(int) | list | List from 0 up to argument–1 |  |
| reduce(function, iterable, any) | any | Iterates a two-argument function over a sequence.  Example: reduce(f, [1,2,3], -1) returns f(f(f(-1, 1), 2), 3).  Example: reduce(f, [1,2,3]) returns f(f(1, 2), 3). |  |
| reversed(iterable) | iterable | Returns a sequence in reverse order |  |
| round(float) | float | Rounds a number to the closest integer |  |
| round(float, int) | float | Rounds a number to the given number of decimal places after the decimal point |  |
| set(iterable) | set | Set constructor |  |
| sin(float) | float | Sine | math |
| sinh(float) | float | Hyperbolic sine | math |
| sorted(iterable) | list | Returns a sequence in sorted order |  |
| sqrt(float) | float | Square root. Fails for negative arguments | math |
| stdev(  TimeSeries) | float | Standard deviation of a time series | cityopt |
| stdev(iterable) | float | Sample standard deviation of a sequence | cityopt |
| str(any) | str | String constructor |  |
| sum(iterable) | number | Sum of sequence elements |  |
| tan(float) | float | Tangent | math |
| tanh(float) | float | Hyperbolic tangent | math |
| timedelta(  float, float) | timedelta | timedelta constructor from number of days, and optionally seconds. | datetime |
| todatetime(  float) | datetime | Converts a simulation time value into a datetime object. Inverse of tosimtime(datetime). | cityopt |
| tosimtime(  datetime) | float | Converts datetime object to a simulation time value. Inverse of todatetime(float). | cityopt |
| tosimtime(  str) | float | Converts ISO-8601 formatted string such as “2015-06-19T21:30:00” to a simulation time value. | cityopt |
| TimeSeries | TimeSeries | TimeSeries constructor | cityopt |
| True | bool | Boolean constant |  |
| tuple(any, …) | tuple | Tuple constructor |  |
| var(TimeSeries) | float | Variance of a time series | cityopt |
| var(iterable) | float | Sample variance of a sequence | cityopt |
| xrange(int) | iterable | Sequence from 0 up to argument – 1. |  |
| zip(iterable, …) | list | Combines given iterables into one iterable of tuples.  Example: zip([1,2,3], [9,8,7]) returns [(1,9), (2,8), (3,7)] |  |
| \_datetime | module | Functions for date and time manipulation | datetime |
| \_\_builtin\_\_ | module | Built-in Python functions | \_\_builtin\_\_ |

## Reserved Python keywords

Python keywords cannot be used as user-defined names. All keywords of Python 2.7 are listed in Table 4. The table includes a description for keywords that can be used in expressions. See the Python language reference for full details [1].

Table 4 Python keywords that cannot be used as user-defined names.

| **Keyword** | **Usage in expressions** |
| --- | --- |
| and | Logical and |
| as |  |
| assert |  |
| break |  |
| class |  |
| continue |  |
| def |  |
| del |  |
| elif |  |
| else | Can be used in conditional expressions.  Example: 9 if 2+3 < 4 else 10 evaluates to 10 |
| except |  |
| exec |  |
| finally |  |
| for | Iteration in generator expressions.  Example: [2\*x for x in [1,2,3]] evaluates to [2,4,6] |
| from |  |
| global |  |
| if | 1. Selection in generator expressions.  Example: [2\*x for x in [1,2,3] if x > 1] evaluates to [4,6]  2. Conditional expressions.  Example: 9 if 2+1 < 4 else 10 evaluates to 9 |
| import |  |
| in | Containment operator.  Example: 2 in [1,2,3] evaluates to True |
| is | Reference equality |
| lambda | Anonymous function definition.  Example: reduce(lambda x,y: 10\*x+y, [1,2,3]) evaluates to 123 |
| not | Logical not |
| or | Logical or |
| pass |  |
| print |  |
| raise |  |
| return |  |
| try |  |
| while |  |
| with |  |
| yield |  |

## Technical details for Python users

The global environment is set up with the following Python statements:

from \_\_future\_\_ import division

import \_\_builtin\_\_, math, cmath, itertools

import cityopt, cityopt.syntax

import datetime as \_datetime

from datetime import \*

from math import \*

del pow

from cityopt import \*

The del statement cancels the import of the float-only pow function from the math module: we prefer the built-in pow function that has special-case support for both integer and float arguments.

The global environment includes more global objects than are reserved as per Table 3; it is nevertheless best to refer to other objects via their module-qualified names such as \_\_builtin\_\_.filter, math.degrees or \_datetime.time.

## References

1. Python 2.7 language reference: <https://docs.python.org/2.7/reference/index.html>
2. Python 2.7 library reference: <https://docs.python.org/2.7/library/index.html>

1. The results of addition and subtraction are exact even in cases where the result is discontinuous: When performing arithmetic between a step function to a piecewise linear function, at each point of discontinuity the left limit value and the right limit value are computed separately. At the discontinuity point, the time series value is equal to the right limit value. [↑](#footnote-ref-2)
2. Since TimeSeries is intentionally not an iterable object, the built-in Python min and max functions are overridden by a CITYOPT specific function that has special support for time series. [↑](#footnote-ref-3)