Prefix expression evaluators

Prefix audio expression evaluator.

This family implements a tiny functional programming language that can be used to write synthesis or signal processing algorithms.

API documentation

This API is in alpha stage and subject to future changes!

Builtin functions

Arithmetic operators

- (+ x y) : returns the sum of two values.
- (- x y) : substracts the second value to the first and returns the result.
- (* x y) : returns the multiplication of two values.
- (/ x y) : returns the quotient of x/y.
- (^ x y) : returns x to the power y.
- (% x y) : returns the floating-point remainder of x/y.
- (neg x) : returns the negative of x.

Moving phase operators

- (++ x y) : increments its internal state by x and wrap around 0.0 and y.
- (– x y) : decrements its internal state by x and wrap around 0.0 and y.
- (~ x y) : generates a periodic ramp from 0 to 1 with frequency x and phase y.

Conditional operators

- (< x y) : returns 1 if x is less than y, otherwise returns 0.
- (<= x y) : returns 1 if x is less than or equal to y, otherwise returns 0.
- (> x y) : returns 1 if x is greater than y, otherwise returns 0.
- (>= x y) : returns 1 if x is greater than or equal to y, otherwise returns 0.
- (== x y) : returns 1 if x is equal to y, otherwise returns 0.
- (!= x y) : returns 1 if x is not equal to y, otherwise returns 0.
- (if (cond) (then) (else)) : returns then for any non-zero value of cond, otherwise returns else.
- (and x y) : returns 1 if both x and y are not 0, otherwise returns 0.
- (or x y) : returns 1 if one of x or y are not 0, otherwise returns 0.

Trigonometric functions

- (sin x) : returns the sine of an angle of x radians.
- (cos x) : returns the cosine of an angle of x radians.

- (tan x) : returns the tangent of x radians.
- (tanh x) : returns the hyperbolic tangent of x radians.
- (atan x) : returns the principal value of the arc tangent of x, expressed in radians.
- (atan2 x y) : returns the principal value of the arc tangent of y/x, expressed in radians.

Power and logarithmic functions

- (sqrt x) : returns the square root of x.
- (log x) : returns the natural logarithm of x.
- (log2 x) : returns the binary (base-2) logarithm of x.
- (log10 x) : returns the common (base-10) logarithm of x.
- (pow x y) : returns x to the power y.

Clipping functions

- (abs x) : returns the absolute value of x.
- (floor x) : rounds x downward, returning the largest integral value that is not greater than x.
- (ceil x) : rounds x upward, returning the smallest integral value that is not less than x.
- (exp x) : returns the constant e to the power x.
- (round x) : returns the integral value that is nearest to x.
- (min x y) : returns the smaller of its arguments: either x or y.
- (max x y) : returns the larger of its arguments: either x or y.
- (wrap x) : wraps x between 0 and 1.

Random fuctions

- (randf x y) : returns a pseudo-random floating-point number in the range between x and y.
- (randi x y) : returns a pseudo-random integral number in the range between x and y.

Filter functions

- (sah x y) : samples and holds x value whenever y is smaller than its previous state.
- (rpole x y) : real one-pole recursive filter. returns x + last_out * y.
- (rzero x y) : real one-zero non-recursive filter. returns x - last_x * y.

Constants

- (const x) : returns x.
- (pi) : returns an approximated value of pi.
- (twopi) : returns a constant with value pi*2.
- (e) : returns an approximated value of e.

Comments

A comment starts with two slashs (//) and ends at the end of the line:

```
// This is a comment!
```

Input and Output signals

User has access to the last buffer size of input and output samples.

To use samples from past input, use $x[n] notation, where n is the position from the current time. $x[0] is the current input, $x[-1] is the previous one and $x[-buffersize] is the last available input sample.

To use samples from past output, use $y[n] notation, where n is the position from the current time. $y[-1] is the previous output and $y[-buffersize] is the last available output sample.

Here an example of a first-order IIR lowpass filter expression:

```
// A first-order IIR lowpass filter
+ $x[0] (* (- $y[-1] $x[0]) 0.99)
```

Defining custom functions

The define keyword starts the definition of a custom function:

```
(define funcname (body))
```

funcname is the name used to call the function in the expression and body is the sequence of functions to execute. Arguments of the function are extracted directly from the body. They must be named $1, $2, $3, ..., $9.

Example of a sine wave function:

```
(define osc (
    sin (* (twopi) (~ $1))
    )
)
// play a sine wave
* (osc 440) 0.3
```

State variables

User can create state variable with the keyword “let”. This is useful to set an intermediate state to be used in multiple places in the processing chain. The syntax is:

```
(let #var (body))
```

The variable name must begin with a “#”:

```
(let #sr 44100)
(let #freq 1000)
```

```
(let #coeff (
    ^ (e) (/ (* (* -2 (pi)) #freq) #sr)
    )
)
+ $x[0] (* (- $y[-1] $x[0]) #coeff)
```

The variable is private to a function if created inside a custom function:

```
(let #freq 250) // global #freq variable
(define osc (
    (let #freq (* $1 $2)) // local #freq variable
    sin (* (twopi) (~ #freq))
    )
)
* (+ (osc 1 #freq) (osc 2 #freq)) 0.2
```

State variables can be used to do 1 sample feedback if used before created. Undefined variables are initialized to 0:

```
(define oscloop (
        (let #xsin
            (sin (+ (* (~ $1) (twopi)) (* #xsin $2))) // #xsin used before...
        ) // ... "let" statement finished!
        #xsin // oscloop function outputs #xsin variable
    )
)
* (oscloop 200 0.7) 0.3
```

User variables

User variables are created with the keyword “var”:

```
(var #var (init))
```

The variable name must begin with a “#”.

They are computed only at initialization, but can be changed from the python script with method calls (varname is a string and value is a float):

```
obj.setVar(varname, value)
```

Library importation

Custom functions can be defined in an external file and imported with the “load” function:

```
(load path/to/the/file)
```

The content of the file will be inserted where the load function is called and all functions defined inside the file will then be accessible. The path can be absolute or relative to the current working directory.

Examples

Here is some expression examples.

A first-order IIR lowpass filter:

```
(var #sr 44100)
(var #cutoff 1000)
(let #coeff (exp (/ (* (* -2 (pi)) #cutoff) #sr)))
+ $x[0] (* (- $y[-1] $x[0]) #coeff)
```

A LFO'ed hyperbolic tangent distortion:

```
// $1 = lfo frequency, $2 = lfo depth
(define lfo (
        (+ (* (sin (* (twopi) (~ $1))) (- $2 1)) $2)
    )
)
tanh (* $x[0] (lfo .25 10))
```

A triangle waveform generator (use Sig(0) as input argument to bypass input):

```
(var #freq 440)
// $1 = oscillator frequency
(define triangle (
        (let #ph (~ $1))
        (- (* (min #ph (- 1 #ph)) 4) 1)
    )
)
triangle #freq
```

Objects

Expr

class **Expr**(*input*, *expr=''*, *mul=1*, *add=0*) [source]

Prefix audio expression evaluator.

Expr implements a tiny functional programming language that can be used to write synthesis or signal processing algorithms.

For documentation about the language, see the module's documentation.

Parent: `PyoObject`

Args: input: PyoObject

Input signal to process.

expr: str, optional

Expression to evaluate as a string.

```
>>> s = Server().boot()
>>> s.start()
>>> proc = '''
>>> (var #boost 1)
>>> (tanh (* $x[0] #boost))
>>> '''
>>> sf = SfPlayer(SNDS_PATH + "/transparent.aif", loop=True)
>>> ex = Expr(sf, proc, mul=0.4).out()
>>> lfo = Sine(freq=1).range(1, 20)
>>> def change():
...     ex.setVar("#boost", lfo.get())
>>> pat = Pattern(change, 0.02).play()
```

setInput(*x*, *fadetime=0.05*) [source]

Replace the *input* attribute.

Args: x: PyoObject

New signal to process.

fadetime: float, optional

Crossfade time between old and new input. Defaults to 0.05.

setExpr(*x*) [source]

Replace the *expr* attribute.

Args: x: string

New expression to process.

printNodes() [source]

Print the list of current nodes.

editor(*title='Expr Editor'*, *wxnoserver=False*) [source]

Opens the text editor for this object.

Args: title: string, optional

Title of the window. If none is provided, the name of the class is used.

wxnoserver: boolean, optional

With wxPython graphical toolkit, if True, tells the interpreter that there will be no server window.

If *wxnoserver* is set to True, the interpreter will not wait for the server GUI before showing the controller window.

input

PyoObject. Input signal to process.

expr

string. New expression to process.