
amalgam-lisp

Release v0.2.0

PureFunctor

Nov 13, 2020

FUNCTIONS TUTORIALS

1	Functions	3
1.1	Arithmetic	3
1.2	Boolean	3
1.3	Control	4
1.4	Comparison	4
1.5	IO	4
1.6	Meta	5
1.7	String	5
1.8	Vector	5
2	Tutorials	7
2.1	Creating Macros	7
3	Amalgams	9
4	Engine	15
5	Environment	17
6	Parser	19
7	Primordials	21
	Python Module Index	25
	Index	27



amalgam lisp

LISP-like interpreted language implemented in Python.

FUNCTIONS

Documentation for the various built-in functions.

1.1 Arithmetic

(+ e0 e1 ... en)

Performs a summation of the provided expressions.

(* e0 e1 ... en)

Performs a multiplication of the provided expressions.

(- e0 e1 ... en)

Subtracts e0 by the sum of e1 ... en.

(/ e0 e1 ... en)

Divides e0 by the product of e1 ... en

1.2 Boolean

(bool e)

Checks for the truthiness of an expression

(not e)

Checks for the falsiness of an expression

(and e0 e1 ... en)

Evaluates and reduces the expressions using an *and* operation, short-circuiting evaluation when a falsy value is encountered.

(or e0 e1 ... en)

Evaluates and reduces the expressions using an *or* operation, short-circuiting evaluation when a truthy value is encountered.

1.3 Control

(if condition then else)

Basic branching construct. If `condition` evaluates to a truthy value, evaluates `then`, otherwise, evaluates `else`.

(when condition then)

Synonym for `if` where `else` defaults to `NIL`.

(cond [[condition value] ...])

Traverses pairs of conditions and values. If the condition evaluates to a truthy value, returns the provided value, short-circuiting evaluation. Returns `NIL` when no conditions are met.

(do expression...)

Evaluates each expression, returning what was evaluated last.

(loop expression...)

Evaluates each expression, looping back at the end. Can be broken by the `return` and `break` functions.

(return expression)

Breaks out of a loop evaluating to a value.

(break)

Breaks out of a loop evaluating to `NIL`.

1.4 Comparison

(> x y)

Performs a *greater than* operation.

(< x y)

Performs a *less than* operation.

(= x y)

Performs a *equal to* operation.

(/= x y)

Performs a *not equal to* operation.

(>= x y)

Performs a *greater than or equal to* operation.

(<= x y)

Performs a *less than or equal to* operation.

1.5 IO

(print expression)

Prints an expression.

(putStrLn string)

Prints the contents of a `string`.

(exit code)

Stops the execution of the program.

1.6 Meta

(setn name value)

Lexically binds a literal name to a value.

(setr name-ref value)

Lexically binds an evaluated name-ref to a value.

(unquote qexpression)

Unquotes a given qexpression.

(eval expression)

Fully evaluates an expression, optionally removing a layer of quoting.

(fn [args...] body)

Creates a scoped lambda given a vector of args, and a body. Binds to a closure when created inside of one.

(mkfn name [args...] body)

Creates a scoped function given a name, a vector of args, and a body. Binds to a closure when created inside of one.

(macro name [args...] body)

Creates a scoped macro given a name, a vector of args, and a body. Binds to a closure when created inside of one.

(let [[name value]...] body)

Creates a closure where each value is bound to a name before evaluating body.

1.7 String

(concat s0 s1 ... sn)

Concatenates strings together.

1.8 Vector

(merge v0 v1 ... vn)

Merges vectors together.

(slice vector start stop step)

Slices a vector using start, stop, and step.

(at index vector)

Returns an item from a vector given its index.

(remove index vector)

Removes an item from a vector given its index.

(len vector)

Returns the length of a vector.

(cons item vector)

Prepends an item to a vector.

(snoc vector item)

Appends an item to a vector.

(is-map vector)

Verifies if the `vector` is a map.

(map-in vector atom)

Checks whether `atom` is a member of `vector`.

(map-at vector atom)

Obtains the value bound to `atom` in `vector`.

(map-up vector atom value)

Sets or updates the `atom` member of `vector` with `atom`.

TUTORIALS

2.1 Creating Macros

Macros are a flexible way to define new language constructs within the language. This tutorial implements a *for-each* loop as an example.

First, let's define the form of our *for-each* loop:

```
(for target in vector expression)
```

`target` is the name of the symbol, `vector` is the sequence we're iterating through, and `expression` being the actions performed for every iteration.

We can use the `macro` function to define this form:

```
(macro for [target in vector expression]
  (do (setn vector (eval vector))
      (setn index 0)
      (setn final (len vector))
      (loop (when (= index final) (break))
            (setr target (at index vector))
            (eval expression)
            (setn index (+ index 1)))))
```

Analysing this example, two procedures are to be observed, specifically, the use of `setr` and `eval`. First, the `vector` that is passed to the macro contains unevaluated expressions that first have to be evaluated using `eval`. As described by the form that we've written earlier, `target` is the name of the symbol that we're assigning to, and as such, we'll have to use `setr` instead of `setn` to make sure that we're not binding using `target` literally. We then evaluate the `expression` literal, which has access to the value bound to `target`.

Let's test it out on the REPL:

```
> (macro for [target in vector expression]
|   (do (setn vector (eval vector))
|       (setn index 0)
|       (setn final (len vector))
|       (loop (when (= index final) (break))
|             (setr target (at index vector))
|             (eval expression)
|             (setn index (+ index 1)))))
for
> (for x in [1 2 3 4 5]
|   (print (* x x)))
1
```

(continues on next page)

(continued from previous page)

```
4  
9  
16  
25  
:NIL
```

The newly defined `for` macro is able to iterate through the vector, binding each number to `x` before evaluating the expression.

Note: Macros create their own closure, and symbols bound within them are inaccessible after evaluation.

Modifying the macro to return the expression evaluated last is left as an exercise for the reader.

AMALGAMS

Internal documentation for the `amalgam.amalgams` module.

class `amalgam.amalgams.Located`

The base dataclass for encapsulating location data of nodes.

Provides an API similar to Lark's `Token` class for convenience.

line_span

Lines spanned by a node

Type `Tuple[int, int]`

column_span

Columns spanned by a node

Type `Tuple[int, int]`

located_on `(*, lines: Tuple[int, int] = (- 1, - 1), columns: Tuple[int, int] = (- 1, - 1)) → amalgam.amalgams.Located`

Helper method for setting `Located.line_span` and `Located.column_span`.

property column

The starting column number of a node.

property end_column

The ending column number of a node.

property end_line

The ending line number of a node.

property line

The starting line number of a node.

class `amalgam.amalgams.Amalgam`

The abstract base class for language constructs.

`__make_repr` `(value: Any) → str`

Helper method for creating a `__repr__()`.

`bind` `(environment: Environment) → Amalgam`

Protocol for implementing environment binding for *Function*.

This base implementation is responsible for allowing `bind` to be called on other *Amalgam* subclasses by performing no operation aside from returning `self`.

`call` `(environment: Environment, *arguments: Amalgam) → Amalgam`

Protocol for implementing function calls for *Function*.

This base implementation is responsible for making the type signature of *SExpression.func* to properly type check when *SExpression.evaluate()* is called, as well as returning a fatal *Notification* for non-callable types.

abstract evaluate (*environment: Environment*) → Any
Protocol for evaluating or unwrapping *Amalgam* objects.

class amalgam.amalgams.**Atom** (*value: str*)
An *Amalgam* that represents different atoms.

value
The name of the atom.

Type str

evaluate (*_environment: Environment*) → *Atom*
Evaluates to the same *Atom* reference.

class amalgam.amalgams.**Numeric** (*value: N*)
An *Amalgam* that wraps around numeric types.
Parameterized as a Generic by: N = TypeVar("N", int, float, Fraction)

value
The numeric value being wrapped.

Type N

evaluate (*_environment: Environment*) → *Numeric*
Evaluates to the same *Numeric* reference.

class amalgam.amalgams.**Symbol** (*value: str*)
An *Amalgam* that wraps around symbols.

value
The name of the symbol.

Type str

evaluate (*environment: Environment*) → *Amalgam*
Searches the provided *environment* fully with *Symbol.value*. Returns the *Amalgam* object bound to the *Symbol.value* in the environment. Returns a fatal *Notification* if a binding is not found.

class amalgam.amalgams.**Function** (*name: str, fn: Callable[...], amalgam.amalgams.Amalgam,*
defer: bool = False, contextual: bool = False)
An *Amalgam* that wraps around functions.

name
The name of the function.

Type str

fn
The function being wrapped. Must have the signature: (*env, amalgams...*) -> *amalgam*.

Type Callable[..., Amalgam]

defer
If set to False, arguments are evaluated before being passed to *Function.fn*.

Type bool

contextual
If set to True, disallows function calls when *Function.in_context* is set to False.

Type bool

env

The *environment.Environment* instance bound to the function. Overrides the *environment* parameter passed to the *Function.call()* method.

Type *environment.Environment*

in_context

Predicate that disallows functions to be called outside of specific contexts. Makes *Function.call()* return a fatal *Notification* when set to False and *Function.contextual* is set to True.

Type bool

bind (*environment: Environment*) → *Function*

Sets the *Function.env* attribute and returns the same *Function* reference.

call (*environment: Environment*, **arguments: Amalgam*) → *Amalgam*

Performs the call to the *Function.fn* attribute.

Performs pre-processing depending on the values of *Function.defer*, *Function.contextual*, and *Function.in_context*,

evaluate (*_environment: Environment*) → *Function*

Evaluates to the same *Function* reference.

with_name (*name: str*) → *amalgam.amalgams.Function*

Sets the *Function.name* attribute and returns the same *Function* reference.

class *amalgam.amalgams.SExpression* (**vals: amalgam.amalgams.Amalgam*)

An *Amalgam* that wraps around S-Expressions.

vals

Entities contained by the S-Expression.

Type Tuple[*Amalgam*, ...]

evaluate (*environment: Environment*) → *Amalgam*

Evaluates *func* using *environment* before invoking the *call()* method with *environment* and *SExpression.args*.

property args

The rest of the *SExpression.vals*.

property func

The head of the *SExpression.vals*.

class *amalgam.amalgams.Vector* (**vals: T*)

An *Amalgam* that wraps around a homogenous vector.

Parameterized as a Generic by: T = TypeVar("T", bound=Amalgam)

vals

Entities contained by the vector

Type Tuple[T, ...]

mapping

Mapping representing vectors with *Atom*s for odd indices and *Amalgam*s for even indices.

Type Mapping[str, *Amalgam*]

_as_mapping () → Mapping[str, *amalgam.amalgams.Amalgam*]

Attempts to create a Mapping[str, *Amalgam*] from *Vector.vals*.

Odd indices must be *Atom*s and even indices must be *Amalgam*s. Returns an empty mapping if this form is not met.

evaluate (*environment*: *Environment*) → *Amalgam*
Creates a new *Vector* by evaluating every value in *Vector.vals*.

class amalgam.amalgams.**Quoted** (*value*: *T*)
An *Amalgam* that defers evaluation of other *Amalgam*s.
Parameterized as a Generic by: *T* = *TypeVar*("T", bound=*Amalgam*)

value
The *Amalgam* being deferred.
Type *T*

evaluate (*_environment*: *Environment*) → *Quoted*
Evaluates to the same *Quoted* reference.

class amalgam.amalgams.**Internal** (*value*: *P*)
An *Amalgam* that holds Python objects.
Parameterized as a Generic by: *P* = *TypeVar*("P", bound=*object*)

value
The Python object being wrapped.
Type *P*

evaluate (*_environment*: *Environment*) → *Internal*
Evaluates to the same *Internal* reference.

class amalgam.amalgams.**Trace** (*amalgam*: *Amalgam*, *environment*: *Environment*, *message*: *str*)
Encapsulates information for tracking notifications.

property amalgam
Alias for field number 0

property environment
Alias for field number 1

property message
Alias for field number 2

class amalgam.amalgams.**Notification** (*, *fatal*: *bool* = *True*, *payload*: *amalgam.amalgams.Amalgam* = *<Atom 'NIL' @ 0x7f2594ecb290>*)
An *Amalgam* that encapsulates and tracks notifications.

fatal
Specifies whether the notification should unconditionally propagate and halt evaluation.
Type *bool*

payload
An optional payload to be carried by a notification.
Type *Amalgam*

trace
A stack of *Trace* objects that tell how the notification propagated.
Type *List*[*Trace*]

evaluate (*_environment*: *Environment*) → *Notification*
Evaluates to the same *Notification* reference.

make_report (*text: str, source: str = '<unknown>'*) → str
Generates a report to be printed to `sys.stderr`.

Accepts `text` and `source` for prettified output.

pop () → *amalgam.amalgams.Trace*
Pops a *Trace* from *Notification.trace*.

push (*amalgam: Amalgam, environment: Environment, message: str*) → None
Pushes a *Trace* into *Notification.trace*.

ENGINE

Internal documentation for the `amalgam.engine` module.

class `amalgam.engine.Engine`

Class that serves as the frontend for parsing and running programs.

environment

An `environment.Environment` instance containing the built-in functions and a reference to the `engine.Engine` instance wrapped within a `amalgams.Internal`, accessible through the `~engine~` key.

Type `environment.Environment`

_interpret (*text: str, source: str = '<unknown>'*) → `amalgam.amalgams.Amalgam`

Parses and runs a text from a source.

Internal-facing method intended for use within `amalgam.primordials`.

interpret (*text: str, source: str = '<unknown>', file: IO = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*) → None

Parses and runs a text from a source.

User-facing method intended for use within `amalgam.cli`. Prints the result to `sys.stdout` unless specified. Handles pretty-printing of `amalgams.Notifications`.

repl (*, *prompt: str = '> ', prompt_cont: str = '| '*) → None

Runs a REPL session that supports multi-line input.

Parameters

- **prompt** (`str`) – The style of the prompt on regular lines.
- **prompt_cont** (`str`) – The style of the prompt on continued lines.

ENVIRONMENT

Internal documentation for the `amalgam.environment` module.

```
class amalgam.environment.Environment (bindings: Bindings = None, parent: Environment =  
                                         None, name: str = 'unknown', engine: Engine =  
                                         None)
```

Class that manages and represents nested execution environments.

bindings

A mapping of `str` keys to `amalgams.Amalgam` values.

Type `Dict[str, Amalgam]`

parent

The parent `Environment` instance to search into, forming a linked list.

Type `Optional[Environment]`

level

The current length of the `Environment` linked list. If a `parent` is provided, sets the current value to the parent's `level` + 1.

Type `int`

search_depth

The search depth when traversing the `Environment` linked list in the `__contains__()`, `__delitem__()`, `__getitem__()`, and `__setitem__()` methods.

Type `int`

name

The name of the execution environment.

Type `str`

engine

A reference to the engine managing the `parser.Parser` instance and the global `Environment` instance.

Type `Engine`

__contains__ (*item: str*) → `bool`

Recursively checks whether an `item` exists.

Searches with respect to the current `search_depth` of the calling `Environment` instance. If the target `item` is encountered at a certain depth less than the target depth, immediately returns `True`, otherwise, returns `False`.

__delitem__ (*item: str*) → `None`

Attempts to recursively delete the provided `item`.

Searches with respect to the current *search_depth* of the calling *Environment* instance. If an existing *item* is encountered at a certain depth less than the target depth, deletes that *item* instead.

__getitem__ (*item: str*) → *Amalgam*

Attempts to recursively obtain the provided *item*.

Searches with respect to the current *search_depth* of the calling *Environment* instance. If an existing *item* is encountered at a certain depth less than the target depth, returns that *item*, otherwise, raises *SymbolNotFound*.

__setitem__ (*item: str, value: Amalgam*) → *None*

Attempts to recursively set the provided *value* to an *item*.

Searches with respect to the current *search_depth* of the calling *Environment* instance. If an existing *item* is encountered at a certain depth less than the target depth, overrides that *item* instead.

env_pop () → *amalgam.environment.Environment*

Discards the current *Environment* and returns the parent *Environment*.

env_push (*bindings: Bindings = None, name: str = None*) → *Environment*

Creates a new *Environment* and binds the calling instance as its parent environment.

search_at (*, *depth=0*)

Context manager for temporarily setting the lookup depth.

The provided *depth* argument must not exceed the *level* of the calling *Environment* instance, and will raise a *ValueError* if done so.

```
>>> env = Environment(FUNCTIONS)
>>>
>>> with env.search_at(depth=42):
...     env["+"] # Raises ValueError
```

Any negative integer can be passed as a *depth* to signify an infinite lookup until the top-most environment.

```
>>> env = Environment(FUNCTIONS)
>>> cl_env = env.env_push({...})
>>>
>>> with cl_env.search_at(depth=-1):
...     cl_env["+"] # Searches `env`
```

class *amalgam.environment.TopLevelPop*

Raised at *Environment.env_pop()*.

PARSER

Internal documentation for the `amalgam.parser` module.

`amalgam.parser.parse` (*text*: str, *source*: str = '<unknown>') → *amalgam.amalgams.Amalgam*
Facilitates regular parsing that can fail.

class `amalgam.parser.Expression` (*visit_tokens*=True)
Transforms expressions in text into their respective *amalgams.Amalgam* representations.

atom (*colon*, *identifier*)

floating (*number*)

fraction (*number*)

integral (*number*)

quoted (*quote*, *expression*)

s_expression (**values*)

string (**values*)

symbol (*identifier*)

vector (**values*)

class `amalgam.parser.ParsingError` (*line*: int, *column*: int, *text*: str, *source*: str)
Base exception for errors during parsing.

line
the line number nearest to the error

Type int

column
the column number nearest to the error

Type int

text
the original text being parsed

Type str

source
the source of the original text

Type str

class `amalgam.parser.ExpectedEOF` (*line*: int, *column*: int, *text*: str, *source*: str)
Raised when multiple expressions are found.

class amalgam.parser.**ExpectedExpression** (*line: int, column: int, text: str, source: str*)
Raised when no expressions are found.

class amalgam.parser.**MissingClosing** (*line: int, column: int, text: str, source: str*)
Raised on missing closing parentheses or brackets.

class amalgam.parser.**MissingOpening** (*line: int, column: int, text: str, source: str*)
Raised on missing opening parentheses or brackets.

PRIMORDIALS

`amalgam.primordials._add(_env: Environment, *nums: am.Numeric) → am.Numeric`
Returns the sum of nums.

`amalgam.primordials._and(env: Environment, *exprs: am.Amalgam) → am.Atom`
Checks the truthiness of the evaluated `exprs` and performs an *and* operation. Short-circuits when `:FALSE` is returned and does not evaluate subsequent expressions.

`amalgam.primordials._at(_env: Environment, index: am.Numeric, vector: am.Vector) → am.Amalgam`
Indexes vector with index.

`amalgam.primordials._bool(_env: Environment, expr: am.Amalgam) → am.Atom`
Checks for the truthiness of an `expr`.

`amalgam.primordials._break(env: Environment) → am.Notification`
Exits a loop with `:NIL`.

`amalgam.primordials._concat(_env: Environment, *strings: am.String) → am.String`
Concatenates the given strings.

`amalgam.primordials._cond(env: Environment, *pairs: am.Vector[am.Amalgam]) → am.Amalgam`
Traverses pairs of conditions and values. If the condition evaluates to `:TRUE`, returns the value pair and short-circuits evaluation. If no conditions are met, `:NIL` is returned.

`amalgam.primordials._cons(_env: Environment, amalgam: am.Amalgam, vector: am.Vector) → am.Vector`
Prepends an amalgam to vector.

`amalgam.primordials._div(_env: Environment, *nums: am.Numeric) → am.Numeric`
Divides `nums[0]` and the product of `nums[1:]`

`amalgam.primordials._do(env: Environment, *exprs: am.Amalgam) → am.Amalgam`
Evaluates a variadic amount of `exprs`, returning the final expression evaluated.

`amalgam.primordials._eq(_env: Environment, x: am.Amalgam, y: am.Amalgam) → am.Atom`
Performs an *equals* comparison.

`amalgam.primordials._eval(env: Environment, amalgam: am.Amalgam) → am.Amalgam`
Evaluates a given amalgam.

`amalgam.primordials._exit(env: Environment, exit_code: am.Numeric = <Numeric '0' @ 0x7f2594c73a50>) → am.Amalgam`
Exits the program with the given `exit_code`.

`amalgam.primordials._fn(env: Environment, args: am.Vector[am.Symbol], body: am.Amalgam) → am.Function`
Creates an anonymous function using the provided arguments.

Binds `env` to the created `amalgams.Function` if a closure is formed.

`amalgam.primordials._ge` (*env*: Environment, *x*: am.Amalgam, *y*: am.Amalgam) → am.Atom
 Performs a *greater than or equal* comparison.

`amalgam.primordials._gt` (*_env*: Environment, *x*: am.Amalgam, *y*: am.Amalgam) → am.Atom
 Performs a *greater than* comparison.

`amalgam.primordials._if` (*env*: Environment, *cond*: am.Amalgam, *then*: am.Amalgam, *else_*: am.Amalgam) → am.Amalgam
 Checks the truthiness of the evaluated *cond*, evaluates and returns *then* if :TRUE, otherwise, evaluates and returns *else_*.

`amalgam.primordials._is_map` (*_env*: Environment, *vector*: am.Vector) → am.Atom
 Verifies whether *vector* is a mapping.

`amalgam.primordials._le` (*env*: Environment, *x*: am.Amalgam, *y*: am.Amalgam) → am.Atom
 Performs a *less than or equal* comparison.

`amalgam.primordials._len` (*_env*: Environment, *vector*: am.Vector) → am.Numeric
 Returns the length of a vector.

`amalgam.primordials._let` (*env*: Environment, *pairs*: am.Vector[am.Vector], *body*: am.Amalgam) → am.Amalgam
 Creates temporary bindings of names to values specified in *pairs* before evaluating *body*.

`amalgam.primordials._loop` (*env*: Environment, **exprs*: am.Amalgam) → am.Amalgam
 Loops through and evaluates *exprs* indefinitely until a *break* or *return* is encountered.

`amalgam.primordials._lt` (*_env*: Environment, *x*: am.Amalgam, *y*: am.Amalgam) → am.Atom
 Performs a *less than* comparison.

`amalgam.primordials._macro` (*env*: Environment, *name*: am.Symbol, *args*: am.Vector[am.Symbol], *body*: am.Amalgam) → am.Amalgam
 Creates a named macro using the provided arguments.

`amalgam.primordials._make_function` (*name*: str, *func*: Optional[Callable[[...], T]] = None, *defer*: bool = False, *contextual*: bool = False, *allows*: Optional[Sequence[str]] = None) → Union[functools.partial, Callable[[...], T]]
 Transforms a given function *func* into a *Function* and stores it inside of the *FUNCTIONS* mapping.

`amalgam.primordials._map_at` (*_env*: Environment, *vector*: am.Vector, *atom*: am.Atom) → am.Amalgam
 Obtains the value bound to *atom* in *vector*.

`amalgam.primordials._map_in` (*_env*: Environment, *vector*: am.Vector, *atom*: am.Atom) → am.Atom
 Checks whether *atom* is a member of *vector*.

`amalgam.primordials._map_up` (*_env*: Environment, *vector*: am.Vector, *atom*: am.Atom, *amalgam*: am.Amalgam) → am.Vector
 Updates the vector mapping with *:data:`atom*, and *amalgam*.

`amalgam.primordials._merge` (*_env*: Environment, **vectors*: am.Vector) → am.Vector
 Merges the given vectors.

`amalgam.primordials._mkfn` (*env*: Environment, *name*: am.Symbol, *args*: am.Vector[am.Symbol], *body*: am.Amalgam) → am.Amalgam
 Creates a named function using the provided arguments.
 Composes `_fn()` and `_setn()`.

`amalgam.primordials._mul` (*_env*: Environment, **nums*: am.Numeric) → am.Numeric
 Returns the product of *nums*.

`amalgam.primordials._ne` (*env: Environment, x: am.Amalgam, y: am.Amalgam*) → `am.Atom`
 Performs a *not equals* comparison.

`amalgam.primordials._not` (*env: Environment, expr: am.Amalgam*) → `am.Atom`
 Checks and negates the truthiness of *expr*.

`amalgam.primordials._or` (*env: Environment, *exprs: am.Amalgam*) → `am.Atom`
 Checks the truthiness of the evaluated *exprs* and performs an *or* operation. Short-circuits when `:TRUE` is returned and does not evaluate subsequent expressions.

`amalgam.primordials._print` (*env: Environment, amalgam: am.Amalgam*) → `am.Amalgam`
 Prints the provided *amalgam* and returns it.

`amalgam.primordials._putstrln` (*env: Environment, string: am.String*) → `am.String`
 Prints the provided *string* and returns it.

`amalgam.primordials._remove` (*env: Environment, index: am.Numeric, vector: am.Vector*) → `am.Vector`
 Removes an item in *vector* using *index*.

`amalgam.primordials._return` (*env: Environment, result: am.Amalgam*) → `am.Notification`
 Exits a context with a result.

`amalgam.primordials._setn` (*env: Environment, name: am.Symbol, amalgam: am.Amalgam*) → `am.Amalgam`
 Binds *name* to the evaluated *amalgam* value in the immediate *env* and returns that value.

`amalgam.primordials._setr` (*env: Environment, rname: am.Amalgam, amalgam: am.Amalgam*) → `am.Amalgam`
 Attempts to resolve *rname* to a `amalgams.Symbol` and binds it to the evaluated *amalgam* in the immediate *env*.

`amalgam.primordials._slice` (*env: Environment, vector: am.Vector, start: am.Numeric, stop: am.Numeric, step: am.Numeric = <Numeric 'I' @ 0xf259435d890>*) → `am.Vector`
 Returns a slice of the given *vector*.

`amalgam.primordials._snoc` (*env: Environment, vector: am.Vector, amalgam: am.Amalgam*) → `am.Vector`
 Appends an *amalgam* to *vector*.

`amalgam.primordials._sub` (*env: Environment, *nums: am.Numeric*) → `am.Numeric`
 Subtracts *nums*[0] and the summation of *nums*[1:].

`amalgam.primordials._unquote` (*env: Environment, qamalgam: am.Quoted[am.Amalgam]*) → `am.Amalgam`
 Unquotes a given *qamalgam*.

`amalgam.primordials._when` (*env: Environment, cond: am.Amalgam, body: am.Amalgam*) → `am.Amalgam`
 Synonym for `_if()` that defaults *else* to `:NIL`.

PYTHON MODULE INDEX

a

`amalgam.primordials`, [21](#)

Symbols

- `__contains__()` (*amalgam.environment.Environment* method), 17
 - `__delitem__()` (*amalgam.environment.Environment* method), 17
 - `__getitem__()` (*amalgam.environment.Environment* method), 18
 - `__setitem__()` (*amalgam.environment.Environment* method), 18
 - `_add()` (*in module amalgam.primordials*), 21
 - `_and()` (*in module amalgam.primordials*), 21
 - `_as_mapping()` (*amalgam.amalgams.Vector* method), 11
 - `_at()` (*in module amalgam.primordials*), 21
 - `_bool()` (*in module amalgam.primordials*), 21
 - `_break()` (*in module amalgam.primordials*), 21
 - `_concat()` (*in module amalgam.primordials*), 21
 - `_cond()` (*in module amalgam.primordials*), 21
 - `_cons()` (*in module amalgam.primordials*), 21
 - `_div()` (*in module amalgam.primordials*), 21
 - `_do()` (*in module amalgam.primordials*), 21
 - `_eq()` (*in module amalgam.primordials*), 21
 - `_eval()` (*in module amalgam.primordials*), 21
 - `_exit()` (*in module amalgam.primordials*), 21
 - `_fn()` (*in module amalgam.primordials*), 21
 - `_ge()` (*in module amalgam.primordials*), 21
 - `_gt()` (*in module amalgam.primordials*), 22
 - `_if()` (*in module amalgam.primordials*), 22
 - `_interpret()` (*amalgam.engine.Engine* method), 15
 - `_is_map()` (*in module amalgam.primordials*), 22
 - `_le()` (*in module amalgam.primordials*), 22
 - `_len()` (*in module amalgam.primordials*), 22
 - `_let()` (*in module amalgam.primordials*), 22
 - `_loop()` (*in module amalgam.primordials*), 22
 - `_lt()` (*in module amalgam.primordials*), 22
 - `_macro()` (*in module amalgam.primordials*), 22
 - `_make_function()` (*in module amalgam.primordials*), 22
 - `_make_repr()` (*amalgam.amalgams.Amalgam* method), 9
 - `_map_at()` (*in module amalgam.primordials*), 22
 - `_map_in()` (*in module amalgam.primordials*), 22
 - `_map_up()` (*in module amalgam.primordials*), 22
 - `_merge()` (*in module amalgam.primordials*), 22
 - `_mkfn()` (*in module amalgam.primordials*), 22
 - `_mul()` (*in module amalgam.primordials*), 22
 - `_ne()` (*in module amalgam.primordials*), 22
 - `_not()` (*in module amalgam.primordials*), 23
 - `_or()` (*in module amalgam.primordials*), 23
 - `_print()` (*in module amalgam.primordials*), 23
 - `_putstrln()` (*in module amalgam.primordials*), 23
 - `_remove()` (*in module amalgam.primordials*), 23
 - `_return()` (*in module amalgam.primordials*), 23
 - `_setn()` (*in module amalgam.primordials*), 23
 - `_setr()` (*in module amalgam.primordials*), 23
 - `_slice()` (*in module amalgam.primordials*), 23
 - `_snoc()` (*in module amalgam.primordials*), 23
 - `_sub()` (*in module amalgam.primordials*), 23
 - `_unquote()` (*in module amalgam.primordials*), 23
 - `_when()` (*in module amalgam.primordials*), 23
- ## A
- Amalgam* (class in *amalgam.amalgams*), 9
 - amalgam()* (*amalgam.amalgams.Trace* property), 12
 - amalgam.primordials* module, 21
 - args()* (*amalgam.amalgams.SExpression* property), 11
 - Atom* (class in *amalgam.amalgams*), 10
 - atom()* (*amalgam.parser.Expression* method), 19
- ## B
- bind()* (*amalgam.amalgams.Amalgam* method), 9
 - bind()* (*amalgam.amalgams.Function* method), 11
 - bindings* (*amalgam.environment.Environment* attribute), 17
- ## C
- call()* (*amalgam.amalgams.Amalgam* method), 9
 - call()* (*amalgam.amalgams.Function* method), 11
 - column* (*amalgam.parser.ParsingError* attribute), 19
 - column()* (*amalgam.amalgams.Located* property), 9
 - column_span* (*amalgam.amalgams.Located* attribute), 9

contextual (*amalgam.amalgams.Function attribute*),
10

D

defer (*amalgam.amalgams.Function attribute*), 10

E

end_column() (*amalgam.amalgams.Located property*), 9

end_line() (*amalgam.amalgams.Located property*), 9

engine (*amalgam.environment.Environment attribute*),
17

Engine (*class in amalgam.engine*), 15

env (*amalgam.amalgams.Function attribute*), 11

env_pop() (*amalgam.environment.Environment method*), 18

env_push() (*amalgam.environment.Environment method*), 18

environment (*amalgam.engine.Engine attribute*), 15

Environment (*class in amalgam.environment*), 17

environment() (*amalgam.amalgams.Trace property*), 12

evaluate() (*amalgam.amalgams.Amalgam method*),
10

evaluate() (*amalgam.amalgams.Atom method*), 10

evaluate() (*amalgam.amalgams.Function method*),
11

evaluate() (*amalgam.amalgams.Internal method*),
12

evaluate() (*amalgam.amalgams.Notification method*), 12

evaluate() (*amalgam.amalgams.Numeric method*),
10

evaluate() (*amalgam.amalgams.Quoted method*), 12

evaluate() (*amalgam.amalgams.SExpression method*), 11

evaluate() (*amalgam.amalgams.Symbol method*), 10

evaluate() (*amalgam.amalgams.Vector method*), 11

ExpectedEOF (*class in amalgam.parser*), 19

ExpectedExpression (*class in amalgam.parser*), 19

Expression (*class in amalgam.parser*), 19

F

fatal (*amalgam.amalgams.Notification attribute*), 12

floating() (*amalgam.parser.Expression method*), 19

fn (*amalgam.amalgams.Function attribute*), 10

fraction() (*amalgam.parser.Expression method*), 19

func() (*amalgam.amalgams.SExpression property*), 11

Function (*class in amalgam.amalgams*), 10

I

in_context (*amalgam.amalgams.Function attribute*),
11

integral() (*amalgam.parser.Expression method*), 19

Internal (*class in amalgam.amalgams*), 12

interpret() (*amalgam.engine.Engine method*), 15

L

level (*amalgam.environment.Environment attribute*),
17

line (*amalgam.parser.ParsingError attribute*), 19

line() (*amalgam.amalgams.Located property*), 9

line_span (*amalgam.amalgams.Located attribute*), 9

Located (*class in amalgam.amalgams*), 9

located_on() (*amalgam.amalgams.Located method*), 9

M

make_report() (*amalgam.amalgams.Notification method*), 12

mapping (*amalgam.amalgams.Vector attribute*), 11

message() (*amalgam.amalgams.Trace property*), 12

MissingClosing (*class in amalgam.parser*), 20

MissingOpening (*class in amalgam.parser*), 20

module
amalgam.primordials, 21

N

name (*amalgam.amalgams.Function attribute*), 10

name (*amalgam.environment.Environment attribute*), 17

Notification (*class in amalgam.amalgams*), 12

Numeric (*class in amalgam.amalgams*), 10

P

parent (*amalgam.environment.Environment attribute*),
17

parse() (*in module amalgam.parser*), 19

ParsingError (*class in amalgam.parser*), 19

payload (*amalgam.amalgams.Notification attribute*),
12

pop() (*amalgam.amalgams.Notification method*), 13

push() (*amalgam.amalgams.Notification method*), 13

Q

Quoted (*class in amalgam.amalgams*), 12

quoted() (*amalgam.parser.Expression method*), 19

R

repl() (*amalgam.engine.Engine method*), 15

S

s_expression() (*amalgam.parser.Expression method*), 19

search_at() (*amalgam.environment.Environment method*), 18

search_depth (*amalgam.environment.Environment attribute*), 17

`SExpression` (class in *amalgam.amalgams*), 11
`source` (*amalgam.parser.ParsingError* attribute), 19
`string()` (*amalgam.parser.Expression* method), 19
`Symbol` (class in *amalgam.amalgams*), 10
`symbol()` (*amalgam.parser.Expression* method), 19

T

`text` (*amalgam.parser.ParsingError* attribute), 19
`TopLevelPop` (class in *amalgam.environment*), 18
`trace` (*amalgam.amalgams.Notification* attribute), 12
`Trace` (class in *amalgam.amalgams*), 12

V

`vals` (*amalgam.amalgams.SExpression* attribute), 11
`vals` (*amalgam.amalgams.Vector* attribute), 11
`value` (*amalgam.amalgams.Atom* attribute), 10
`value` (*amalgam.amalgams.Internal* attribute), 12
`value` (*amalgam.amalgams.Numeric* attribute), 10
`value` (*amalgam.amalgams.Quoted* attribute), 12
`value` (*amalgam.amalgams.Symbol* attribute), 10
`Vector` (class in *amalgam.amalgams*), 11
`vector()` (*amalgam.parser.Expression* method), 19

W

`with_name()` (*amalgam.amalgams.Function* method),
11