amalgam-lisp

Release v0.1.0

PureFunctor

Nov 03, 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 **13**

5 Environment **15**

6 Parser **17**

7 Primordials **19**

Python Module Index **25**

Index **27**

LISP-like interpreted language implemented in Python.

CHAPTER

ONE

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

(require module-path)
Loads a module given its `module-path`, importing the exposed symbols to the current environment with respect to the export list created by `provide`.

(provide symbols...)
Creates an export list of `symbols` when loaded through `require`.

(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 [qargs...] body)
Creates a scoped macro given a `name`, a vector of `qargs`, 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`.

CHAPTER

TWO

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 [qtarget in qvector qexpression] ...)
```

`target`, `vector`, and `expression` are prefixed with `q` as arguments to macros are passed literally as quoted values that are not evaluated, giving the macro the ability to selective unquote and evaluate expressions.

Let's now define what happens within the macro:

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

The values being manipulated in the macro are first unquoted and bound to names without their `q` prefix:

```
(do (setn target (unquote qtarget))
    (setn vector (unquote qvector))
    (setn expression (unquote qexpression)))
```

Then, the looping logic is defined:

```
(setn index 0)
(setn final (len vector))
(loop (when (= index final) (break))
```

(continues on next page)

(continued from previous page)

```
      (setr target (at index vector))
      (eval expression)
      (setn index (+ index 1)))
```

Two things are to be observed here, specifically, the use of `setr` and `eval`. As described by the form that we've defined 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 to `target` literally. We then evaluate the `expression` literal, which then has access to the value bound to `target`.

Let's test it out on the REPL:

```
> (macro for [qtarget in qvector qexpression]
|        (do (setn target (unquote qtarget))
|            (setn vector (unquote qvector))
|            (setn expression (unquote qexpression))
|            (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
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.

CHAPTER

THREE

AMALGAMS

Internal documentation for the `amalgam.amalgams` module.

class `amalgam.amalgams.`**`Amalgam`**

The abstract base class for language constructs.

`_make_repr`(*value: Any*) → str

Helper method for creating a `__repr__()`.

`bind`(*environment:* amalgam.environment.Environment) → *amalgam.amalgams.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:* amalgam.environment.Environment, **arguments:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.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 raising `NotImplementedError` for non-callable types.

`abstract evaluate`(*environment:* amalgam.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:* amalgam.environment.Environment) → *amalgam.amalgams.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:* amalgam.environment.Environment) → *amalgam.amalgams.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:* amalgam.environment.Environment) → *amalgam.amalgams.Amalgam*

Searches the provided *environment* fully with `Symbol.value`. Returns the `Amalgam` object bound to the `Symbol.value` in the environment. Raises `environment.SymbolNotFound` 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 `True`, arguments are wrapped in `Quoted` 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()` raise `DisallowedContextError` when set to `False` and `Function.contextual` is set to `True`.

Type `bool`

bind(*environment:* amalgam.environment.Environment) → *amalgam.amalgams.Function*

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

call(*environment:* amalgam.environment.Environment, **arguments:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.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:* amalgam.environment.Environment) → *amalgam.amalgams.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:* amalgam.environment.Environment) → *amalgam.amalgams.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*]

Attemps 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:* amalgam.environment.Environment) → *amalgam.amalgams.Vector*

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:* amalgam.environment.Environment) → *amalgam.amalgams.Quoted*

Evaluates to the same *Quoted* reference.

class amalgam.amalgams.**Internal**(*value: P*)

An *Amalgam* that holds Python `object` s.

Parameterized as a `Generic` by: `P = TypeVar("P", bound=object)`

value

The Python `object` being wrapped.

Type `P`

evaluate(*_environment:* amalgam.environment.Environment) → *amalgam.amalgams.Internal*

Evaluates to the same `Internal` reference.

class amalgam.amalgams.**DisallowedContextError**

Raised on functions outside of their intended contexts.

CHAPTER

FOUR

ENGINE

Internal documentation for the `amalgam.engine` module.

class `amalgam.engine.`**`Engine`**

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

`parser`

A `parser.Parser` instance.

Type `parser.Parser`

`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`

`parse_and_run`(*text: str*) → *amalgam.amalgams.Amalgam*

Parses and runs the given *text* string.

`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.

CHAPTER

FIVE

ENVIRONMENT

Internal documentation for the `amalgam.environment` module.

class `amalgam.environment.`**`Environment`**(*bindings: Bindings = None, parent: Environment = 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`

`__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*) → *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.**SymbolNotFound**

Synonym for `KeyError`.

class amalgam.environment.**TopLevelPop**

Raised at `Environment.env_pop()`.

CHAPTER

SIX

PARSER

Internal documentation for the `amalgam.parser` module.

class `amalgam.parser.`**`Parser`**

Class that serves as the frontend for parsing text.

`parse_buffer`

The text buffer used within `Parser.repl_parse()`.

Type `StringIO`

`parse`(*text: str*) → *amalgam.amalgams.Amalgam*

Facilitates regular parsing that can fail.

`repl_parse`(*text: str*) → Optional[*amalgam.amalgams.Amalgam*]

Facilitates multi-line parsing for the REPL.

Writes the given *text* string to the `parse_buffer` and attempts to parse *text*.

If `MissingClosing` is raised, returns *None* to allow for parsing to continue.

If another subclass of `ParsingError` is raised, clears the `parse_buffer` and re-raises the exception.

Otherwise, if parsing succeeds, clears the `parse_buffer` and returns the parsed expression.

class `amalgam.parser.`**`Expression`**(*visit_tokens=True*)

Transforms expressions in text into their respective `amalgams.Amalgam` representations.

`atom`(*identifier: lark.lexer.Token*) → *amalgam.amalgams.Atom*

`floating`(*number: lark.lexer.Token*) → *amalgam.amalgams.Numeric*

`fraction`(*number: lark.lexer.Token*) → *amalgam.amalgams.Numeric*

`integral`(*number: lark.lexer.Token*) → *amalgam.amalgams.Numeric*

`quoted`(*expression:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Quoted*

`s_expression`(**expressions:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.SExpression*

`string`(**values: lark.lexer.Token*) → amalgam.amalgams.String

`symbol`(*identifier: lark.lexer.Token*) → *amalgam.amalgams.Symbol*

`vector`(**expressions:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Vector*

class `amalgam.parser.`**`ParsingError`**(*line: int*, *column: int*)

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`

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

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

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

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

CHAPTER

SEVEN

PRIMORDIALS

`amalgam.primordials.`**`_add`**(*_env:* amalgam.environment.Environment, **nums:* amalgam.amalgams.Numeric) → *amalgam.amalgams.Numeric*

Returns the sum of `nums`.

`amalgam.primordials.`**`_and`**(*env:* amalgam.environment.Environment, **qexprs:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Atom*

Checks the truthiness of the evaluated `qexprs` and performs an *and* operation. Short-circuits when `:FALSE` is returned and does not evaluate subsequent expressions.

`amalgam.primordials.`**`_at`**(*_env:* amalgam.environment.Environment, *index:* amalgam.amalgams.Numeric, *vector:* amalgam.amalgams.Vector) → *amalgam.amalgams.Amalgam*

Indexes `vector` with `index`.

`amalgam.primordials.`**`_bool`**(*_env:* amalgam.environment.Environment, *expr:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Checks for the truthiness of an `expr`.

`amalgam.primordials.`**`_break`**(*env:* amalgam.environment.Environment) → *amalgam.amalgams.Internal*

Exits a loop with `:NIL`.

`amalgam.primordials.`**`_concat`**(*_env:* amalgam.environment.Environment, **strings:* *amalgam.amalgams.String*) → amalgam.amalgams.String

Concatenates the given `strings`.

`amalgam.primordials.`**`_cond`**(*env:* amalgam.environment.Environment, **qpairs:* amalgam.amalgams.Quoted[amalgam.amalgams.Vector[amalgam.amalgams.Amalgam]]) → *amalgam.amalgams.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:* amalgam.environment.Environment, *amalgam:* amalgam.amalgams.Amalgam, *vector:* amalgam.amalgams.Vector) → *amalgam.amalgams.Vector*

Preprends an `amalgam` to `vector`.

`amalgam.primordials.`**`_div`**(*_env:* amalgam.environment.Environment, **nums:* amalgam.amalgams.Numeric) → *amalgam.amalgams.Numeric*

Divides `nums[0]` and the product of `nums[1:]`

`amalgam.primordials.`**`_do`**(*env:* amalgam.environment.Environment, **qexprs:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Evaluates a variadic amount of `qexprs`, returning the final expression evaluated.

`amalgam.primordials._eq`(*_env:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs an *equals* comparison.

`amalgam.primordials._eval`(*env:* amalgam.environment.Environment, *amalgam:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Amalgam*

Evaluates a given `amalgam`.

`amalgam.primordials._exit`(*env:* *amalgam.environment.Environment*, *exit_code:* *amalgam.amalgams.Numeric = <Numeric '0' @ 0x7f8b3babbbd0>*) → *amalgam.amalgams.Amalgam*

Exits the program with the given `exit_code`.

`amalgam.primordials._fn`(*env:* amalgam.environment.Environment, *args:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Vector*[*amalgam.amalgams.Symbol*]]*, *body:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*) → *amalgam.amalgams.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:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs a *greater than or equal* comparison.

`amalgam.primordials._gt`(*_env:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs a *greater than* comparison.

`amalgam.primordials._if`(*env:* amalgam.environment.Environment, *qcond:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*, *qthen:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*, *qelse:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*) → *amalgam.amalgams.Amalgam*

Checks the truthiness of the evaluated `qcond`, evaluates and returns `qthen` if `:TRUE`, otherwise, evaluates and returns `qelse`.

`amalgam.primordials._is_map`(*_env:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector) → *amalgam.amalgams.Atom*

Verifies whether `vector` is a mapping.

`amalgam.primordials._le`(*env:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs a *less than or equal* comparison.

`amalgam.primordials._len`(*_env:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector) → *amalgam.amalgams.Numeric*

Returns the length of a `vector`.

`amalgam.primordials._let`(*env:* amalgam.environment.Environment, *qpairs:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Vector*[*amalgam.amalgams.Vector*]]*, *body:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*) → *amalgam.amalgams.Amalgam*

Creates temporary bindings of names to values specified in `qpairs` before evaluating `body`.

`amalgam.primordials._loop`(*env:* amalgam.environment.Environment, **qexprs:* amalgam.amalgams.Quoted*[*amalgam.amalgams.Amalgam*]*) → *amalgam.amalgams.Amalgam*

Loops through and evaluates `qexprs` indefinitely until a `break` or `return` is encountered.

`amalgam.primordials._lt`(*_env:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs a *less than* comparison.

`amalgam.primordials._macro`(*env:* amalgam.environment.Environment, *name:* amalgam.amalgams.Quoted[amalgam.amalgams.Symbol], *args:* amalgam.amalgams.Quoted[amalgam.amalgams.Vector[amalgam.amalgams.Symbol]], *body:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Creates a named macro using the provided arguments.

`amalgam.primordials._make_function`(*name: str*, *func: Callable[[...], T] = None*, *defer: bool = False*, *contextual: bool = False*, *allows: 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:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector, *atom:* amalgam.amalgams.Atom) → *amalgam.amalgams.Amalgam*

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

`amalgam.primordials._map_in`(*_env:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector, *atom:* amalgam.amalgams.Atom) → *amalgam.amalgams.Atom*

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

`amalgam.primordials._map_up`(*_env:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector, *atom:* amalgam.amalgams.Atom, *amalgam:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Vector*

Updates the `vector mapping with :data:`atom`, and `amalgam`.

`amalgam.primordials._merge`(*_env:* amalgam.environment.Environment, **vectors:* amalgam.amalgams.Vector) → *amalgam.amalgams.Vector*

Merges the given `vectors`.

`amalgam.primordials._mkfn`(*env:* amalgam.environment.Environment, *name:* amalgam.amalgams.Quoted[amalgam.amalgams.Symbol], *args:* amalgam.amalgams.Quoted[amalgam.amalgams.Vector[amalgam.amalgams.Symbol]], *body:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Creates a named function using the provided arguments.

Composes *`_fn()`* and *`_setn()`*.

`amalgam.primordials._mul`(*_env:* amalgam.environment.Environment, **nums:* amalgam.amalgams.Numeric) → *amalgam.amalgams.Numeric*

Returns the product of `nums`.

`amalgam.primordials._ne`(*_env:* amalgam.environment.Environment, *x:* amalgam.amalgams.Amalgam, *y:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Performs a *not equals* comparison.

`amalgam.primordials._not`(*_env:* amalgam.environment.Environment, *expr:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Atom*

Checks and negates the truthiness of `expr`.

`amalgam.primordials._or`(*env:* amalgam.environment.Environment, **qexprs:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Atom*

Checks the truthiness of the evaluated `qexprs` and performs an *or* operation. Short-circuits when `:TRUE` is returned and does not evaluate subsequent expressions.

`amalgam.primordials.`**`_print`**(*_env:* amalgam.environment.Environment, *amalgam:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Amalgam*

Prints the provided `amalgam` and returns it.

`amalgam.primordials.`**`_provide`**(*env:* amalgam.environment.Environment, **qsymbols:* amalgam.amalgams.Quoted[amalgam.amalgams.Symbol]) → *amalgam.amalgams.Atom*

Sets the *~provides~* key to be used in `_require()`.

`amalgam.primordials.`**`_putstrln`**(*_env:* amalgam.environment.Environment, *string:* *amalgam.amalgams.String*) → amalgam.amalgams.String

Prints the provided `string` and returns it.

`amalgam.primordials.`**`_remove`**(*_env:* amalgam.environment.Environment, *index:* amalgam.amalgams.Numeric, *vector:* amalgam.amalgams.Vector) → *amalgam.amalgams.Vector*

Removes an item in `vector` using `index`.

`amalgam.primordials.`**`_require`**(*env:* amalgam.environment.Environment, *module_name:* *amalgam.amalgams.String*) → *amalgam.amalgams.Atom*

Runs a given `module_name` and imports the exposed symbols to the current `env` with respect to the *~provides~* key created in `_provide()`.

`amalgam.primordials.`**`_return`**(*env:* amalgam.environment.Environment, *result:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Internal*

Exits a context with a `result`.

`amalgam.primordials.`**`_setn`**(*env:* amalgam.environment.Environment, *name:* amalgam.amalgams.Quoted[amalgam.amalgams.Symbol], *amalgam:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Binds `name` to the evaluated `amalgam` value in the immediate `env` and returns that value.

`amalgam.primordials.`**`_setr`**(*env:* amalgam.environment.Environment, *qrname:* amalgam.amalgams.Quoted[amalgam.amalgams.Symbol], *qamalgam:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Attemps to resolve `qrname` to a `amalgams.Symbol` and binds it to the evaluated `qamalgam` in the immediate `env`.

`amalgam.primordials.`**`_slice`**(*_env:* *amalgam.environment.Environment*, *vector:* *amalgam.amalgams.Vector*, *start:* *amalgam.amalgams.Numeric*, *stop:* *amalgam.amalgams.Numeric*, *step:* *amalgam.amalgams.Numeric* = *<Numeric '1' @ 0x7f8b3bac3250>*) → *amalgam.amalgams.Vector*

Returns a slice of the given `vector`.

`amalgam.primordials.`**`_snoc`**(*_env:* amalgam.environment.Environment, *vector:* amalgam.amalgams.Vector, *amalgam:* amalgam.amalgams.Amalgam) → *amalgam.amalgams.Vector*

Appends an `amalgam` to `vector`.

`amalgam.primordials.`**`_sub`**(*_env:* amalgam.environment.Environment, **nums:* amalgam.amalgams.Numeric) → *amalgam.amalgams.Numeric*

Subtracts `nums[0]` and the summation of `nums[1:]`.

`amalgam.primordials.`**`_unquote`**(*_env:* amalgam.environment.Environment, *qamalgam:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Unquotes a given `qamalgam`.

`amalgam.primordials.`**`_when`**(*env:* amalgam.environment.Environment, *qcond:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam], *qbody:* amalgam.amalgams.Quoted[amalgam.amalgams.Amalgam]) → *amalgam.amalgams.Amalgam*

Synonym for *_if()* that defaults `qelse` to `:NIL`.

PYTHON MODULE INDEX

a

amalgam.primordials, 19

INDEX

Symbols

__contains__() (*amalgam.environment.Environment method*), 15
__delitem__() (*amalgam.environment.Environment method*), 15
__getitem__() (*amalgam.environment.Environment method*), 15
__setitem__() (*amalgam.environment.Environment method*), 15
_add() (*in module amalgam.primordials*), 19
_and() (*in module amalgam.primordials*), 19
_as_mapping() (*amalgam.amalgams.Vector method*), 11
_at() (*in module amalgam.primordials*), 19
_bool() (*in module amalgam.primordials*), 19
_break() (*in module amalgam.primordials*), 19
_concat() (*in module amalgam.primordials*), 19
_cond() (*in module amalgam.primordials*), 19
_cons() (*in module amalgam.primordials*), 19
_div() (*in module amalgam.primordials*), 19
_do() (*in module amalgam.primordials*), 19
_eq() (*in module amalgam.primordials*), 19
_eval() (*in module amalgam.primordials*), 20
_exit() (*in module amalgam.primordials*), 20
_fn() (*in module amalgam.primordials*), 20
_ge() (*in module amalgam.primordials*), 20
_gt() (*in module amalgam.primordials*), 20
_if() (*in module amalgam.primordials*), 20
_is_map() (*in module amalgam.primordials*), 20
_le() (*in module amalgam.primordials*), 20
_len() (*in module amalgam.primordials*), 20
_let() (*in module amalgam.primordials*), 20
_loop() (*in module amalgam.primordials*), 20
_lt() (*in module amalgam.primordials*), 21
_macro() (*in module amalgam.primordials*), 21
_make_function() (*in module amalgam.primordials*), 21
_make_repr() (*amalgam.amalgams.Amalgam method*), 9
_map_at() (*in module amalgam.primordials*), 21
_map_in() (*in module amalgam.primordials*), 21
_map_up() (*in module amalgam.primordials*), 21
_merge() (*in module amalgam.primordials*), 21
_mkfn() (*in module amalgam.primordials*), 21
_mul() (*in module amalgam.primordials*), 21
_ne() (*in module amalgam.primordials*), 21
_not() (*in module amalgam.primordials*), 21
_or() (*in module amalgam.primordials*), 21
_print() (*in module amalgam.primordials*), 22
_provide() (*in module amalgam.primordials*), 22
_putstrln() (*in module amalgam.primordials*), 22
_remove() (*in module amalgam.primordials*), 22
_require() (*in module amalgam.primordials*), 22
_return() (*in module amalgam.primordials*), 22
_setn() (*in module amalgam.primordials*), 22
_setr() (*in module amalgam.primordials*), 22
_slice() (*in module amalgam.primordials*), 22
_snoc() (*in module amalgam.primordials*), 22
_sub() (*in module amalgam.primordials*), 22
_unquote() (*in module amalgam.primordials*), 22
_when() (*in module amalgam.primordials*), 22

A

Amalgam (*class in amalgam.amalgams*), 9
amalgam.primordials module, 19
args() (*amalgam.amalgams.SExpression property*), 11
Atom (*class in amalgam.amalgams*), 9
atom() (*amalgam.parser.Expression method*), 17

B

bind() (*amalgam.amalgams.Amalgam method*), 9
bind() (*amalgam.amalgams.Function method*), 10
bindings (*amalgam.environment.Environment attribute*), 15

C

call() (*amalgam.amalgams.Amalgam method*), 9
call() (*amalgam.amalgams.Function method*), 10
column (*amalgam.parser.ParsingError attribute*), 17
contextual (*amalgam.amalgams.Function attribute*), 10

D

defer (*amalgam.amalgams.Function attribute*), 10
DisallowedContextError (*class in amalgam.amalgams*), 12

E

Engine (*class in amalgam.engine*), 13
env (*amalgam.amalgams.Function attribute*), 10
env_pop() (*amalgam.environment.Environment method*), 16
env_push() (*amalgam.environment.Environment method*), 16
environment (*amalgam.engine.Engine attribute*), 13
Environment (*class in amalgam.environment*), 15
evaluate() (*amalgam.amalgams.Amalgam method*), 9
evaluate() (*amalgam.amalgams.Atom method*), 9
evaluate() (*amalgam.amalgams.Function method*), 10
evaluate() (*amalgam.amalgams.Internal method*), 11
evaluate() (*amalgam.amalgams.Numeric method*), 9
evaluate() (*amalgam.amalgams.Quoted method*), 11
evaluate() (*amalgam.amalgams.SExpression method*), 11
evaluate() (*amalgam.amalgams.Symbol method*), 10
evaluate() (*amalgam.amalgams.Vector method*), 11
ExpectedEOF (*class in amalgam.parser*), 18
ExpectedExpression (*class in amalgam.parser*), 18
Expression (*class in amalgam.parser*), 17

F

floating() (*amalgam.parser.Expression method*), 17
fn (*amalgam.amalgams.Function attribute*), 10
fraction() (*amalgam.parser.Expression method*), 17
func() (*amalgam.amalgams.SExpression property*), 11
Function (*class in amalgam.amalgams*), 10

I

in_context (*amalgam.amalgams.Function attribute*), 10
integral() (*amalgam.parser.Expression method*), 17
Internal (*class in amalgam.amalgams*), 11

L

level (*amalgam.environment.Environment attribute*), 15
line (*amalgam.parser.ParsingError attribute*), 17

M

mapping (*amalgam.amalgams.Vector attribute*), 11
MissingClosing (*class in amalgam.parser*), 18
MissingOpening (*class in amalgam.parser*), 18
module
    amalgam.primordials, 19

N

name (*amalgam.amalgams.Function attribute*), 10
Numeric (*class in amalgam.amalgams*), 9

P

parent (*amalgam.environment.Environment attribute*), 15
parse() (*amalgam.parser.Parser method*), 17
parse_and_run() (*amalgam.engine.Engine method*), 13
parse_buffer (*amalgam.parser.Parser attribute*), 17
parser (*amalgam.engine.Engine attribute*), 13
Parser (*class in amalgam.parser*), 17
ParsingError (*class in amalgam.parser*), 17

Q

Quoted (*class in amalgam.amalgams*), 11
quoted() (*amalgam.parser.Expression method*), 17

R

repl() (*amalgam.engine.Engine method*), 13
repl_parse() (*amalgam.parser.Parser method*), 17

S

s_expression() (*amalgam.parser.Expression method*), 17
search_at() (*amalgam.environment.Environment method*), 16
search_depth (*amalgam.environment.Environment attribute*), 15
SExpression (*class in amalgam.amalgams*), 10
string() (*amalgam.parser.Expression method*), 17
Symbol (*class in amalgam.amalgams*), 9
symbol() (*amalgam.parser.Expression method*), 17
SymbolNotFound (*class in amalgam.environment*), 16

T

TopLevelPop (*class in amalgam.environment*), 16

V

vals (*amalgam.amalgams.SExpression attribute*), 11
vals (*amalgam.amalgams.Vector attribute*), 11
value (*amalgam.amalgams.Atom attribute*), 9
value (*amalgam.amalgams.Internal attribute*), 11
value (*amalgam.amalgams.Numeric attribute*), 9
value (*amalgam.amalgams.Quoted attribute*), 11
value (*amalgam.amalgams.Symbol attribute*), 10
Vector (*class in amalgam.amalgams*), 11
vector() (*amalgam.parser.Expression method*), 17

W

`with_name()` (*amalgam.amalgams.Function method*), 10