Package ‘easier’

August 6, 2013

Type Package

Version 0.1-1

Date 2013-05-29

Title A collection of Python-esque data types

Author Greg Lamp and Austin Ogilvie

Maintainer Greg Lamp <greg@yhathq.com>

Depends R (>= 2.12.0)

Imports plyr, digest, methods

Suggests testthat (>= 0.2)

Description easier provides Python-like data types (list and dict) in R

License FreeBSD

URL https://github.com/yhat/easier

BugReports https://github.com/yhat/easier/issues

Collate ’pydict.R’ ’pylist.R’ ’utils.r’

R topics documented:

dict.py *Creates an instance of a dict*

Description

This is a wrapper function around the `pydict$new` that is a little more R friendly.

Usage

```
dict.py(...)
```

Arguments

`...`	a series of key/value pairs in the form `key=value`

Examples

```
(x <- dict.py("a"=1, "b"=2, "c"=3))
#{a: 1, b: 2, c: 3}
```

dict_repl *Function for representing hashed objects as strings*

Description

Purely visual.

Usage

```
dict_repl(object, obj_name)
```

Arguments

`object`	an arbitrary thing
`obj_name`	name of the variable as defined by the user (not currently being used)

`encapsulate` *Helper function for making character vectors have quotes around each item when printed to the console.*

Description

Helper function for making character vectors have quotes around each item when printed to the console.

Usage

```
encapsulate(values)
```

Arguments

`values`	a vector of values

`hist` *Plots a histogram of the items of a list.*

Description

Generic function that plots a histogram of the items in a list.

`is.dict.py` *Determines whether or not an object is an instance of a dictionary.*

Description

Determines the class of an object and checks to see if it's a dictionary.

Usage

```
is.dict.py(object)
```

Arguments

`object`	any object

Examples

```
x <- dict.py("a"=1)
is.dict.py(x)
#TRUE
x <- list(1, 2, 3, 4)
is.dict.py(x)
#FALSE
```

is.list.py — *Determines whether or not an object is an instance of a list*

Description

Determines the class of an object and checks to see if it's a list

Usage

```
is.list.py(object)
```

Arguments

`object`	any object

Examples

```
x <- list.py("a")
is.list.py(x)
#TRUE
x <- 1:10
is.dict.py(x)
#FALSE
```

lapply — *Wrapper around* `lapply`.

Description

Automatically invotes `lapply` on the items in the list.

list.py — *Creates an instance of a list*

Description

This is a wrapper function around the `pylist$new` that is a little more R friendly.

Usage

```
list.py(...)
```

Arguments

`...`	a series of values seperated by a comma. NOTE: a vector will be treated as an individual item. i.e. `list.py(1:100)` will yield a list with 1 item, whereas `list.py(1, 2, 3, 4)` will yield a list with 4 items

Examples

```
x <- list.py(1, 2, 3, 4)
#[1, 2, 3, 4]
```

`merge.list` *Function that takes 2 lists and merges them fairly effeciently*

Description

Function that takes 2 lists and merges them fairly effeciently

Usage

```
merge.list(x, y = NULL, mergeUnnamed = TRUE, ...)
```

Arguments

`x`	a list
`y`	a second list
`mergeUnnamed`	boolean for whether or not to include list items with no names
`...`	whatever else you've got

`paste` *Turns a list into a printable string*

Description

Generic function that calls the toString method for a list.

`plot` *Plots a scatterplot of the items of a list.*

Description

Generic function that plots a scatterplot of the items in a list.

`sapply` *Wrapper around* `sapply`.

Description

Automatically invotes `sapply` on the items in the list.

`summary` *Creates a summary of the items in a list.*

Description

Sumamrizes the list by data type. Each data type gets it’s own summary with the results put into a native R list.

`toString` *Turns a list into a string.*

Description

Generic function that calls the string method for a list.

`zip.dict` *Combine 2 lists into a dict of key/values*

Description

Takes 2 lists and converts them into a key => value mapping, which takes the form of a `dict.py`.

Usage

```
zip.dict(x, y)
```

Arguments

x	a list, vector, or list.py
y	a second list, vector, or list.py

Examples

```
x <- list.py(1, 2, 3)
y <- list.py("a", "b", "c")
zip.dict(x, y)
#{1: 'a', 2: 'b', 3: 'c'}
zip.dict(y, x)
#{'a': 1, 'b': 2, 'c': 3}
```

zip.tuple *Combine 2 lists into a list of lists*

Description

Return a list of 2 item lists, where each list contains the i-th element from each of the argument sequences. The returned list is truncated in length to the length of the shortest argument sequence.

Usage

```
zip.tuple(x, y)
```

Arguments

x	a list, vector, or list.py
y	a second list, vector, or list.py

Examples

```
x <- list.py(1, 2, 3)
y <- list.py(4, 5, 6)
zip.tuple(x, y)
#[[1, 4], [2, 5], [3, 6]]
y <- list.py("a", "b", "c")
zip.tuple(x, y)
#[[1, 'a'], [2, 'b'], [3, 'c']]
```

[*Get the value of a key associated with a dictionary.*

Description

You can use the adict['key'] syntax to access key/values from within a dictionary–much like Python, Ruby, or Perl.

You can use the adict['key'] syntax to set key/values from within a dictionary–much like Python, Ruby, or Perl.

Use much like length(list()) or length(c(1, 2, 3)).

You can use the adict[idx] syntax to access items from within a list–much like Python, Ruby, or Perl.

You can use the alist[idx] syntax to set items within a list–much like Python, Ruby, or Perl.

Generic function that calls the string method for a list.

Generic function for caclulating the sum of the items in a list. If an item is not numeric an error occurs.

Generic function for caclulating the cumsum of the items in a list. If an item is not numeric an error occurs.

Generic function for caclulating the sin of the items in a list. If an item is not numeric an error occurs.

Generic function for caclulating the cos of the items in a list. If an item is not numeric an error occurs.

Generic function for caclulating the sign of the items in a list. If an item is not numeric an error occurs.

Use much like length(list(1, 2, 3)) or length(c(1, 2, 3)).

Arguments

`x`	a list
`...`	named args

Examples

```
as.character(list.py(1, 2, 3, 4))
```

Index