import-python-web

Release 1.0.4

Christopher James

May 27, 2022

CONTENTS

import-python-web - Python's built-in functions, classes and (some) modules ported to JavaScript. Most are pretty close to the same functionality as the python version of the function or class.

Completed modules:

- core (base python without importing)
- random
- itertools
- string

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Usage

1.1.1 Installation

```
npm install import-python-web
```

1.1.2 Quickstart

```
const { py_import_star, py_import } = require('import-python-web')
const itertools = py_import('itertools') // import a module
const { randint } = py_import('random') // destructured import

// import all Python core built-in function and classes into the global namespace
py_import_star('core')

// use any Python core function or class as a JavaScript function
from (let i of range(10, 20, 2)) {
   print(list([i, randint(10, 20)]), end=', ')
}
```

1.2 import-python-web

py_import(*name*)

import a python module

Arguments

- **name** (`string()`) – the name of the module

Returns **Object** – the module

```
// import the module to a single variable
const itertools = py_import('itertools')

// destructuring assignment to import the different functions or classes
const { permutations, combinations } = py_import('itertools')
```

py_import_star(*name*)

import all functions and classes from a module into the current namespace

Arguments

- **name** (string()) – the module

```
// only recommended for the core python module but can be used for any module
py_import_star('core')
```

1.3 Core

1.3.1 Classes

class Complex(*real*, *imaginary*)

Class to represent a complex number system (real and imaginary)

Arguments

- **real** (number()) – the real number
- **imaginary** (number()) – the imaginary number

Returns **Complex** –

Complex.**magnitude**

Gets the magnitude of the complex number.

Complex.**phase**

Gets the phase of the complex number.

Complex.**polar**

Gets the polar coordinates of the complex number.

Complex.**add**(*other*)

Add two complex numbers.

Arguments

- **other** (*Complex()*) – the complex number to compare

Returns **Complex** –

Complex.**conjugate**()

Conjugate a complex number.

Returns **Complex** –

Complex.**div**(*other*)

Divide two complex numbers.

Arguments

- **other** (*Complex()*) – the complex number to compare

Returns **Complex** –

Complex.**mul**(*other*)

Multiply two complex numbers.

Arguments

- **other** (*Complex()*) – the complex number to compare

Returns **Complex** –

Complex.**sub**(*other*)

Subtract two complex numbers.

Arguments

- **other** (*Complex()*) – the complex number to compare

Returns **Complex** –

Complex.**toString**()

Return the complex number as a string in the format (a + bi).

Returns **string** –

Complex.**fromPolar**(*magnitude*, *phase*)

Create a complex number from polar coordinates.

Arguments

- **magnitude** (number()) – the magnitude
- **phase** (number()) – the phase

Returns **Complex** –

Complex.**fromString**(*str*)

Create a complex number from a string (a + bi).

Arguments

- **str** (string()) – the string to parse

Returns **Complex** –

class Tuple(*iterable*)

Class representing Python's Tuple

Creates a tuple from an iterable.

Arguments

- **iterable** (iterable()) – the iterable

Throws Error() –

- if the iterable is not iterable

Returns **Tuple** –

Tuple.**count**(*num*)

Counts the number of occurrences of an item in the tuple.

Arguments

- **num** (*any()*) – the item to count

Returns **number** –

Tuple.**index**(*num*)

Returns the index of the first occurrence of an item in the tuple.

Arguments

- **num** (*any()*) – the item to find

Returns **number** –

Tuple.**toString**()

Returns a string representation of the tuple.

Returns **string** –

class List(*iterable*)

Class representing Python's List

Creates a list from an iterable.

Arguments

- **iterable** (iterable()) – the iterable

Throws Error() –

- if the iterable is not iterable

Returns **List** –

List.**append**(*x*)

Appends an item to the end of the list.

Arguments

- **x** (*any()*) – the item to append

List.**clear**()

Clears the list of all items.

List.**copy**()

Returns a copy of the list.

Returns **List** –

List.**count**(*x*)

Counts the number of occurrences of an item in the list.

Arguments

- **x** (*any()*) – the item to count

Returns **number** –

List.**extend**(*iterable*)

Extends the list with an iterable.

Arguments

- **iterable** (iterable()) – the iterable

Returns **List** –

`List.`**`index`**(*x*, *start=0*, *stop*)

Returns the index of the first occurrence of an item in the list.

Arguments

- **`x`** (*any()*) – the item to find
- **`start`** (`number()`) – the index to start searching from
- **`stop`** (`number()`) – the index to end searching at

Throws

- `Error()` –
  - if the index is out of range
- `Error()` –
  - if the value is not found

Returns **number** –

`List.`**`insert`**(*index*, *x*)

Inserts an item at the specified index.

Arguments

- **`index`** (`number()`) – the index to insert at
- **`x`** (*any()*) – the item to insert

Throws `Error()` –

- if the index is out of range

`List.`**`pop`**(*index*)

removes the last index of the list and returns it. If an index is specified, it removes the item at that index and returns it.

Arguments

- **`index`** (`number()`) – the index to remove

Throws `Error()` –

- if the index is out of range

Returns **any** –

`List.`**`remove`**(*x*)

Removes the first occurrence of an item in the list.

Arguments

- **`x`** (*any()*) – the item to remove

`List.`**`sort`**(*key*, *reverse=false*)

Sort the list in place. If a key function is specified, the list is sorted according to the key function. If a key function is not specified, the list is sorted according to the value of the items. If a reverse flag is specified, the list is sorted in reverse order.

Arguments

- **`key`** (`function()`) – the key function
- **`reverse`** (`boolean()`) – the reverse flag

List.**toString**()

Returns a string representation of the list.

Returns **string** –

class Dict(*iterable*)

Class to represent Python's dictionary

Creates a dictionary from a list of key-value pairs.

Arguments

- **iterable** (iterable|Object()) – iterable or object containing the keys and values grouped together.

Returns **Dict** –

Dict.**clear**()

Clears the dictionary of all key-value pairs.

Dict.**copy**()

Returns a copy of the dictionary.

Returns **Dict** –

Dict.**get**(*key*)

Gets the value of a key.

Arguments

- **key** (string()) – the key

Returns **any** –

Dict.**items**()

Returns the key-value pairs of the dictionary as a list of tuples.

Returns **List** –

Dict.**keys**()

Returns the keys of the dictionary.

Returns **Tuple** –

Dict.**pop**(*key*)

Removes a key from the dictionary. Returns the value of the key.

Arguments

- **key** (string()) – the key

Returns **any** –

Dict.**popitem**()

Removes the last key-value pair from the dictionary and returns it.

Returns **Tuple** –

Dict.**setItem**(*key*, *value*)

Sets the key-value pair.

Arguments

- **key** (string()) – the key

- **value** (*any()*) – the value

Dict.**toString**()

Returns a string representation of the dictionary.

Returns **string** –

Dict.**update**(*other*)

Updates the dictionary with the key-value pairs of another dictionary.

Arguments

- **other** (*Dict()*) – the dictionary to update with

Dict.**values**()

Returns the values of the dictionary.

Returns **Tuple** –

Dict.**fromkeys**(*keys*, *value=null*)

Creates a dictionary from a list of keys using a default value.

Arguments

- **keys** (iterable()) – the keys
- **value** (*any()*) – the default value

Returns **Dict** –

class FrozenSet(*iterable*)

Class representing Python's FrozenSet

Creates a frozen set from an iterable.

Arguments

- **iterable** (iterable()) – the iterable

Throws Error() –

- if the iterable is not iterable

Returns **FrozenSet** –

FrozenSet.**toString**()

Returns a string representation of the frozen set.

Returns **string** –

class String()

Additional methods for strings.

String.**capitalize**()

Convert the first character of the string to uppercase

Returns **string** – the string with the first character converted to uppercase

String.**center**(*width*, *fillchar*)

Return a centered string of length width.

Arguments

- **width** (number()) – the length of the string to be returned
- **fillchar** (string()) – the string to be used for padding

Throws

- `TypeError()` – if width is not an integer
- `Error()` – if width is less than zero

Returns **string** – the centered string

`String.`**count**(*sub*[, *start*[, *end*]])

Return the number of non-overlapping occurrences of substring sub in the range [start, end].

Arguments

- **sub** – substring to count
- **start** – start index of the range
- **end** – end index of the range

Returns number of occurrences

Return type number

`String.`**encode**(*encoding*)

Encode the string using the codec registered for encoding. Default encoding is 'utf-8'.

Arguments

- **encoding** (`string()`) – the encoding to be used

Throws

- `TypeError()` – if encoding is not a string
- `Error()` – if encoding is not a valid encoding

Returns **string** – the encoded string

`String.`**endswith**(*suffix*, *start*, *end*)

Return true if the string ends with the suffix, otherwise return false. If suffix is not a string, it is converted to one using str(suffix). If the optional start, end, or both are supplied, then return true if the string ends with the suffix between start and end positions (including the end position, but not the start position).

Arguments

- **suffix** (`string()`) – the suffix to be checked
- **start** (`number()`) – the start index
- **end** (`number()`) – the end index

Returns **boolean** – true if the string ends with the suffix

`String.`**expandtabs**(*tabsize*)

Return a copy of the string where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 2 characters is assumed.

Arguments

- **tabsize** (`number()`) – the tab size

Throws

- `TypeError()` – if tabsize is not an integer
- `Error()` – if tabsize is less than zero

Returns **string** – the string with tabs expanded

String.**find**(*sub*, *start*, *end*)

Return the lowest index in the string where substring sub is found, such that sub is contained in the slice s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

Arguments

- **sub** (string()) – the substring to be found
- **start** (number()) – the start index
- **end** (number()) – the end index

Throws

- TypeError() – if start is not an integer
- TypeError() – if end is not an integer
- Error() – if start or end is less than zero or end > start

Returns **number** – the index of the substring

String.**format**([*values*])

Return a formatted version of the string using the format string. The format string may contain literal text or replacement fields. The fields are identified by braces {fieldNumber}. Each replacement field contains one or more format specifiers, which define how the corresponding value is converted to a string. The field number is optional, but may be present in the specifiers. If the field number is not present, the fields are filled in in the order they appear in the format string. If field numbers are present, they are used to order the fields in the format string.

Arguments

- **values** – values to insert into the format string

Returns formatted string

Return type string

String.**format_map**(*map*)

Return a formatted version of the string using the format map. The format map should be a Dict or Map but can also be an iterable. of key-value pairs. The fields are identified by braces { } and must contain names that match the keys in the format map.

Arguments

- **map** (Map|Dict()) – the format map

Throws

- TypeError() – if map is not a Map, Dict or iterable of key-value pairs
- Error() – if map is empty
- Error() – if there is an empty specifier in the format string
- Error() – if there is a mismatch between the number of arguments and the number of specifiers

Returns **string** – the formatted string

String.**index**(*sub*[, *start*[, *end*]])

Return the index of the first occurrence of substring sub in the range [start, end].

Arguments

- **sub** – substring to find

- **start** – start index of the range
- **end** – end index of the range

Returns index of the first occurrence

Return type number

String.**isalnum**()

Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are alphanumeric and there is at least one character, false otherwise

String.**isalpha**()

Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are alphabetic and there is at least one character, false otherwise

String.**isdigit**()

Return true if all characters in the string are digits, false otherwise.

Returns **boolean** – true if all characters in the string are digits, false otherwise

String.**isidentifier**()

A string is considered a valid identifier if it only contains alphanumeric letters (a-z) and (0-9), or underscores (_). A valid identifier cannot start with a number, or contain any spaces.

Returns **boolean** – true if the string is a valid identifier, false otherwise

String.**islower**()

Return true if all characters in the string are lowercase and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are lowercase and there is at least one character, false otherwise

String.**isnumeric**()

Return true if all characters in the string are numeric and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are numeric and there is at least one character, false otherwise

String.**isprintable**()

Return true if all characters in the string are printable, false otherwise.

Returns **boolean** – true if all characters in the string are printable, false otherwise

String.**isspace**()

Return true if all characters in the string are whitespace and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are whitespace and there is at least one character, false otherwise

String.**istitle**()

Return true if the string is a titlecased string, false otherwise.

Returns **boolean** – true if the string is a titlecased string, false otherwise

String.**isupper**()

Return true if all characters in the string are uppercase and there is at least one character, false otherwise.

Returns **boolean** – true if all characters in the string are uppercase and there is at least one character, false otherwise

`String.`**`join`**`(`*iterable*`)`

Join the elements of an iterable to the end of the string.

Arguments

- **`iterable`** (`Iterable()`) – the iterable to join

Throws

- `TypeError()` – if iterable is not iterable
- `TypeError()` – if iterable contains a non-string

Returns **string** – the joined string

`String.`**`jsReplace`**`()`

JS's original string replace() method

`String.`**`ljust`**`(`*width*, *fillchar*=`'\" \"'`)`

Return a left-justified version of the string, padding on the right with the specified fill character.

Arguments

- **`width`** (`number()`) – the minimum width of the resulting string
- **`fillchar`** (`string()`) – the fill character

Throws `TypeError()` – if width is not a number

Returns **string** – the left-justified string

`String.`**`lower`**`()`

Return a lowercased version of the string.

Returns **string** – the lowercased string

`String.`**`lstrip`**`(`*chars*=`'\" \"'`)`

Return a left-stripped version of the string.

Arguments

- **`chars`** (`string()`) – the characters to strip

Throws `TypeError()` – if chars is not a string

Returns **string** – the left-stripped string

`String.`**`maketrans`**`(`*from*=`'\'\"'`, *to*=`'\'\"'`)`

Return a translation table to be used in a str.translate() method.

Arguments

- **`from`** (`string()`) – the characters to replace
- **`to`** (`string()`) – the replacement characters

Throws

- `TypeError()` – if from is not a string
- `TypeError()` – if to is not a string
- `TypeError()` – if from and to are not the same length

Returns **Dict** – the translation table

`String.`**`partition`**`(`*sep*`)`

Return a tuple containing the string itself, followed by the first occurrence of sep, and the remainder of the string.

Arguments

- **sep** (`string()`) – the separator

Throws

- `TypeError()` – if sep is not a string
- `ValueError()` – if sep is empty

Returns **Tuple** – the tuple containing the string itself, followed by the first occurrence of sep, and the remainder of the string

`String.`**`replace`**`(`*old*, *new*, *count*`)`

Return a copy of the string with all occurrences of substring old replaced by new.

Arguments

- **old** (`string()`) – the substring to replace
- **new** (`string()`) – the replacement substring
- **count** (`number()`) – the maximum number of occurrences to replace

Returns **string** – the copy of the string with all occurrences of substring old replaced by new

`String.`**`rfind`**`(`*sub*, *start=0*, *end=-1*`)`

Return the highest index in the string where substring sub is found, starting at the end.

Arguments

- **sub** (`string()`) – the substring to find
- **start** (`number()`) – the index to start the search
- **end** (`number()`) – the index to end the search

Throws

- `TypeError()` – if sub is not a string
- `TypeError()` – if start is not a number
- `TypeError()` – if end is not a number
- `ValueError()` – if start is not in the range [0, len(string)]

Returns **number** – the highest index in the string where substring sub is found, starting at the end

`String.`**`rindex`**`(`*sub*, *start=0*, *end=-1*`)`

Return the highest index in the string where substring sub is found, starting at the end.

Arguments

- **sub** (`string()`) – the substring to find
- **start** (`number()`) – the index to start the search
- **end** (`number()`) – the index to end the search

Throws

- `TypeError()` – if sub is not a string
- `TypeError()` – if start is not a number
- `TypeError()` – if end is not a number
- `ValueError()` – if start is not in the range [0, len(string)]

Returns **number** – the highest index in the string where substring sub is found, starting at the end

`String.`**`rjust`**`(`*width*, *fillchar='\" \"'*`)`

Return the string right justified in a string of length width.

Arguments

- **`width`** (`number()`) – the length of the resulting string
- **`fillchar`** (`string()`) – the character to pad the string with

Throws

- `TypeError()` – if width is not a number
- `TypeError()` – if fillchar is not a string
- `ValueError()` – if width is less than zero

Returns **string** – the string right justified in a string of length width

`String.`**`rpartition`**`(`*sep*`)`

Return a tuple containing the string itself, followed by the last occurrence of sep, and the remainder of the string.

Arguments

- **`sep`** (`string()`) – the separator

Throws

- `TypeError()` – if sep is not a string
- `ValueError()` – if sep is empty
- `ValueError()` – if sep is not in the string

Returns **Tuple** – the tuple containing the string itself, followed by the first occurrence of sep, and the remainder of the string

`String.`**`rsplit`**`(`*sep='\" \"'*, *maxsplit=-1*`)`

Return a list of the words in the string, using sep as the delimiter string starting from the right.

Arguments

- **`sep`** (`string()`) – the delimiter string
- **`maxsplit`** (`number()`) – the maximum number of splits

Throws

- `TypeError()` – if sep is not a string
- `TypeError()` – if maxsplit is not a number
- `ValueError()` – if maxsplit is less than zero

Returns **Array.<string>** – the list of the words in the string, using sep as the delimiter string

String.**rstrip**(*chars='\"\''*)

Return a copy of the string with trailing whitespace removed.

Arguments

- **chars** (string()) – the characters to be stripped

Throws

- TypeError() – if chars is not a string
- ValueError() – if chars is empty
- ValueError() – if chars is not in the string

Returns **string** – a copy of the string with trailing whitespace removed

String.**startswith**(*prefix*, *start=0*)

Return true if the string starts with the prefix, otherwise return false.

Arguments

- **prefix** (string()) – the prefix
- **start** (number()) – the index to start searching from

Returns **boolean** – true if the string starts with the prefix, otherwise return false

String.**strip**(*chars='\"\''*)

Return a copy of the string with leading and trailing whitespace removed.

Arguments

- **chars** (string()) – the characters to be stripped

Returns **string** – a copy of the string with leading and trailing whitespace removed

String.**jsSplit**()

The JS implementatino of split()

String.**split**(*sep='\"\''*, *maxsplit=-1*)

splits the string into a list of words using sep as the delimiter string and the maxsplit as the maximum number of splits.

Arguments

- **sep** (string()) – the delimiter string
- **maxsplit** (number()) – the maximum number of splits

Returns **List** –

String.**splitlines**(*keepends=false*)

Return a list of the lines in the string, breaking at line boundaries.

Arguments

- **keepends** (boolean()) – if true, retain line breaks in the resulting list

Throws TypeError() – if keepends is not a boolean

Returns **Array.<string>** – the list of the lines in the string, breaking at line boundaries

`String.swapcase()`

Return a copy of the string with uppercase characters converted to lowercase and vice versa.

Throws `ValueError()` – if the string is empty

Returns **string** – a copy of the string with uppercase characters converted to lowercase and vice versa

`String.title()`

Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

Throws

- `ValueError()` – if the string is empty
- `ValueError()` – if the string contains only whitespace

Returns **string** – a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase

`String.translate(`*table*`='\'\"')`

Return a copy of the string in which each character has been mapped through the given translation table. You can use String.maketrans() to create a translation map from character-to-character mappings in different formats.

Arguments

- **table** (`string()`) – the mapping table

Throws `TypeError()` – if table is not a Dict or Map

Returns **string** – a copy of the string where all characters have been mapped to their uppercase equivalent

`String.upper()`

Return a copy of the string where all characters have been mapped to their uppercase equivalent.

Throws `ValueError()` – if the string is empty

Returns **string** – a copy of the string where all characters have been mapped to their uppercase equivalent

`String.zfill(`*width*`)`

Return a string of length width padded with zeros on the left.

Arguments

- **width** (`number()`) – the length of the resulting string

Throws

- `TypeError()` – if width is not a number
- `ValueError()` – if width is not a positive integer
- `ValueError()` – if the string is empty

Returns **string** – a string of length width padded with zeros on the left

1.3.2 Functions

abs(*x*)

Return the absolute value of a number. The argument may be an integer, a floating point number.

Arguments

- **x** (number()) – The number to take the absolute value of.

Returns **number** – - The absolute value of x.

all(*array*)

Return True if all elements of the iterable are true.

Arguments

- **array** (iterable()) – The iterable to check.

Returns **boolean** –

any(*array*)

Return True if any element of the iterable is true.

Arguments

- **array** (iterable()) – The iterable to check.

Returns **boolean** –

ascii(*obj*)

return a string containing a printable representation of an object, but escape the non-ASCII characters in the string.

Arguments

- **obj** (Object()) – the object

Returns **string** –

bin(*x*)

Convert an integer number to a binary string

Arguments

- **x** (number()) – the number to convert

Returns **string** –

bool(*x*)

Returns a boolean. X is converted to a boolean Using a truth test.

Arguments

- **x** (Object()) – the object to convert

Returns **boolean** –

breakpoint()

This function is used to pause the debugger.

bytearray(*data*)

Return a new array of bytes. The Uint8Array is a typed array that is used to store the bytes and is a mutable sequence of integers in the range 0 to 255.

Arguments

- **data** (iterable()) – The iterable to convert to a Uint8Array.

Returns **Uint8Array** –

bytes(*data*)

Return a tuple containing an immutable sequence of integers from 0 to 255.

Arguments

- **data** (iterable()) – The iterable to convert to a byte array.

Returns **Tuple** –

callable(*obj*)

Return true if the given object is a function or class and false otherwise.

Arguments

- **obj** (Object()) – The object to check.

Returns **boolean** –

chr(*i*)

Return a string containing a character whose Unicode code is the integer i. This is the inverse of ord().

Arguments

- **i** (number()) – The integer to convert to a character.

Returns **string** –

complex(*x*, *i=0*)

Return a complex number with the given real and imaginary parts. If a string is given, it must be in the form "r [+-] xi". Ex: complex("2 + 3i").

Arguments

- **x** (number|string()) – The real part of the complex number or a string in the form "r [+-] xi".
- **i** (number()) – The imaginary part of the complex number.

Returns **Complex** –

delattr(*obj*, *name*)

Delete an attribute from an object. If the attribute is a method, it is removed from the object's method table.

Arguments

- **obj** (Object()) – The object to delete the attribute from.
- **name** (string()) – The name of the attribute to delete.

dict(*iterable*)

Return a dictionary created from the given iterable. The iterable must be an iterable containing two-element iterables representing key-value pairs.

Arguments

- **iterable** (iterable()) – The iterable to create a dictionary from.

Returns **Dict** –

dir(*obj*)

Return a list of names in the given object.

Arguments

- **obj** (`Object()`) – The object to get the names from. Defaults to the global object.

Returns **Array** –

divmod(*num*, *den*)

Return a tuple containing the integer division of x and y and the remainder.

Arguments

- **num** (`number()`) – The dividend.
- **den** (`number()`) – The divisor.

Returns **Tuple** – - The quotient and remainder.

enumerate(*array*, *start=0*, *step=1*)

Return an enumerate object. It contains a pair (index, value) for each item in the array. This is a generator object that returns a tuple containing the index and value of each item.

Arguments

- **array** (`iterable()`) – The iterable to enumerate.
- **start** (`number()`) – The starting index. Defaults to 0.
- **step** (`number()`) – The step. Defaults to 1.

Returns **Tuple** – The enumerate object.

exec(*code*, *globals*, *locals*)

Execute the given code in a new context. The code must be a string.

Arguments

- **code** (`string()`) – The code to execute.
- **globals** (`Object()`) – The global object. Defaults to the global object.
- **locals** (`Object()`) – The local object. Defaults to an empty object.

Returns **Object** – The result of the code.

filter(*func*, *iterable*)

Return a list of those items in iterable for which function(item) is true.

Arguments

- **func** (`function()`) – The function to filter by.
- **iterable** (`iterable()`) – The iterable to filter.

Returns **Array** –

float(*x*)

Return a floating point number constructed from a string.

Arguments

- **x** (`string()`) – The string to convert to a floating point number.

Returns **number** –

format(*string*, **args*)

Return a formatted string. The string must have at least one replacement sequence, otherwise a TypeError is raised.

The embedded sequences are substituted by the values in args. The sequence has the form `{}`.

The result is computed by replacing each occurrence of the embedded sequence with the string representation of the corresponding value.

Example:

```
>>> format("{} {}", "a", "b")
'a b'
```

frozenset(*iterable*)

Returns a new frozen set containing the items in iterable.

Arguments

- **iterable** (`iterable()`) – The iterable to freeze.

Returns **FrozenSet** –

getattr(*obj*, *name*, *defaultValue*)

Return the value of the named attribute of object. If the attribute is not found, default is returned if provided, otherwise AttributeError is raised.

Arguments

- **obj** (`Object()`) – The object to get the attribute from.
- **name** (`string()`) – The name of the attribute to get.
- **defaultValue** (*any()*) – The default value to return if the attribute is not found.

Throws `Error()` – If the attribute is not found and no default value is provided.

Returns **any** –

hasattr(*obj*, *name*)

Return True if the named attribute is found in the given object, otherwise False.

Arguments

- **obj** (`Object()`) – The object to check the attribute in.
- **name** (`string()`) – The name of the attribute to check.

Returns **boolean** –

hex(*x*)

Convert an integer to a hexadecimal string.

Arguments

- **x** (`number()`) – The integer to convert to a hexadecimal string.

Returns **string** –

int(*x*, *base=10*)

Return the integer value of x. If base is given, the string x is first converted to a number in base base. If the string cannot be converted to an integer, a TypeError is raised.

Arguments

- **x** (`string()`) – The string to convert to an integer.
- **base** (`number()`) – The base to convert the string to. Defaults to 10.

Returns **number** –

`isinstance`(*x*, *Type*)

Return True if the given object is an instance of the given type.

Arguments

- **x** (*any()*) – The object to check.
- **Type** (*type()*) – The type to check against.

Returns **boolean** –

`issubclass`(*x*, *type*)

Return True if the given object is a subclass of the given type.

Arguments

- **x** (*any()*) – The object to check.
- **type** (*type()*) – The type to check against.

Returns **boolean** –

`iter`(*obj*)

Return an iterator over the given iterable.

Arguments

- **obj** (`iterable()`) – The iterable to iterate over.

Returns **Iterator** –

`len`(*array*)

Return the length of the given object.

Arguments

- **array** (`iterable()`) – The object to get the length of.

Returns **number** –

`list`(*iterable*)

Create a list containing the items in iterable.

Arguments

- **iterable** (`iterable()`) – The iterable to create a list from.

Returns **List** –

`locals`()

return the global objects.

Returns **Object** –

`map`(*func*, *iterable*)

Return a new list containing the items returned by applying the given function to the items of the given iterable.

Arguments

- **func** (`function()`) – The function to apply to each item.

- **iterable** (iterable()) – The iterable to map over.

Returns **Array** –

max(*array*)

Returns the maximum value in the given iterable.

Arguments

- **array** (iterable()) – The iterable to get the maximum value from.

Returns **number** –

min(*array*)

Return the minimum value in the given iterable.

Arguments

- **array** (iterable()) – The iterable to get the minimum value from.

Returns **number** –

next(*iterator*, *def*)

Return the next item from the iterator.

Arguments

- **iterator** (Iterator()) – The iterator to get the next item from.
- **def** (*any()*) – The default value to return if the iterator is exhausted.

Returns **any** –

oct(*x*)

Return the octal representation of an integer.

Arguments

- **x** (number()) – The integer to convert to an octal string.

Returns **string** –

ord(*x*)

Return the Unicode code point for the given character.

Arguments

- **x** (string()) – The character to get the Unicode code point for.

Returns **number** –

pow(*num*, *exp*)

Return x to the power of y.

Arguments

- **num** (number()) – The base.
- **exp** (number()) – The exponent.

Returns **number** –

print(*text*, *...args*)

Print the given object to the console *no end or sep abilities in web version*.

Arguments

- **text** (*any()*) – The object to print.
- **args** (*any()*) – Any additional objects to print.

range(*stop*, *start=null*, *step=1*)

Return an iterator that produces a range of integers.

Arguments

- **stop** (number()) – The stop value of the range(exclusive). Will be the start value if the start value is given.
- **start** (number()) – The start value of the range(inclusive). Defaults to 0.
- **step** (number()) – The step value of the range. Defaults to 1.

Returns **Iterator** –

repr(*obj*)

Returns a string containing the printable representation of the given object.

Arguments

- **obj** (*any()*) – The object to convert to a string.

Returns **string** –

reversed(*iterable*)

Reverse the order of the given iterable.

Arguments

- **iterable** (iterable()) – The iterable to reverse.

Returns **iterable** –

round(*num*, *ndigits=0*)

Return the rounded value of x to the given number of decimal places.

Arguments

- **num** (number()) – The number to round.
- **ndigits** (number()) – The number of decimal places to round to.

Returns **number** –

set(*array*)

Create a set from the given iterable.

Arguments

- **array** (iterable()) – The iterable to create a set from.

Returns **Set** –

setattr(*obj*, *name*, *value*)

Set the value of an attribute of an object. If the attribute is not present, it will be added.

Arguments

- **obj** (object()) – The object to set the attribute on.
- **name** (string()) – The name of the attribute to set.
- **value** (*any()*) – The value to set the attribute to.

slice(*array*, *start*, *stop*, *step=1*)

Return a slice of the given iterable.

Arguments

- **array** (`iterable()`) – The iterable to slice.
- **start** (`number()`) – The start index of the slice.
- **stop** (`number()`) – The stop index of the slice.
- **step** (`number()`) – The step value of the slice.

Returns **iterable** –

sorted(*iterable*, *key*)

Return a sorted list of the given iterable.

Arguments

- **iterable** (`iterable()`) – The iterable to sort.
- **key** (`function()`) – The function to use to sort the iterable.

Returns **iterable** –

str(*x*)

Return the string representation of the given object.

Arguments

- **x** (*any()*) – The object to convert to a string.

Returns **string** –

sum(*array*)

Return the sum of the given iterable.

Arguments

- **array** (`iterable()`) – The iterable to sum.

Returns **number** –

tuple(*iterable*)

Return a tuple of the given iterable.

Arguments

- **iterable** (`iterable()`) – The iterable to create a tuple from.

Returns **Tuple** –

type(*obj*)

Return the type of the given object.

Arguments

- **obj** (*any()*) – The object to get the type of.

Returns **string** –

zip(*...arrays*)

Return a list of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.

Arguments

- **arrays** (*any()*) – The iterables to zip.

Returns **iterable** –

1.4 Random

class Random(*seed=null*)

Class to generate random numbers.

Create a random number.

Arguments

- **seed** (`number()`) –

Returns **Random** – a random number generator

`Random.`**genRandom**`()`

Generate a random number using the mulberry32 algorithm.

Returns **number** – a random number between 0 and 1

`Random.`**setSeed**(*seed*)

Set the seed of the random number generator.

Arguments

- **seed** (`number|string|boolean()`) –

Throws `Error()` – if the seed is not a number, string, or boolean

Returns **number** – the seed

seed(*seed*)

Set the seed of the random number generator.

Arguments

- **seed** (`number|string|boolean()`) – the seed

Returns **number** – the seed

random`()`

Return a random number between 0 and 1.

Returns **number** – a random number between 0 and 1

randint(*start*, *end*)

Generate a random integer between start and end.

Arguments

- **start** (`number()`) – the start of the range (inclusive)
- **end** (`number()`) – the end of the range (inclusive)

Returns **number** – a random number between start and end

randrange(*stop*, *start=null*, *step=1*)

Return a random integer between start and end with optional step.

Arguments

- **stop** (number()) – the stop of the range (exclusive)
- **start** (number()) – the start of the range (inclusive)
- **step** (number()) – the step between numbers

Returns **number** – a random number between start and stop

choice(*iterable*)

Return a random element from a non-empty iterable.

Arguments

- **iterable** (iterable()) – the iterable

Returns **object** – a random element from the iterable

choices(*iterable*, *k*)

Return k random elements from a non-empty iterable.

Arguments

- **iterable** (iterable()) – the iterable
- **k** (number()) – the number of elements to return

Returns **List** – a list of k random elements from the iterable

sample(*iterable*, *k=1*)

Return k elements from a non-empty iterable without replacement.

Arguments

- **iterable** (iterable()) – the iterable
- **k** (number()) – the number of elements to return

Returns **List** – a list of k elements from the iterable

shuffle(*iterable*)

Shuffle an array in place.

Arguments

- **iterable** (iterable()) – the iterable to shuffle

Returns **List** – a shiffled list of the elements of the iterable

uniform(*start*, *end*)

Return a number between start and end.

Arguments

- **start** (number()) – the start of the range (inclusive)
- **end** (number()) – the end of the range (inclusive)

Returns **number** – a random number between start and end

1.5 Itertools

accumulate(*iterable*, *func*, *initial=null*)

Add the elements of an iterable (or something that can be converted to an iterable) to produce a single value.

Arguments

- **iterable** (`Iterable()`) – The iterable to accumulate over.
- **func** (`function()`) – The function to use to combine the elements.
- **initial** (`number()`) – The initial value to start with.

Returns **Generator** – - A generator that yields the accumulated value.

chain(*...iterables*)

Chain together the results of several iterables.

Arguments

- **iterables** (`Iterable()`) – The iterables to chain together.

Returns **Generator** – - A generator that yields the results of the chained iterables.

combinations(*iterable*, *r*)

Return successive r-length combinations of elements in the iterable.

Arguments

- **iterable** (`Iterable()`) – The iterable to combine over.
- **r** (`number()`) – The number of elements to combine.

Returns **Generator** – - A generator that yields the combinations.

combinations_with_replacement(*iterable*, *r*)

Return successive r-length combinations of elements in the iterable. The combinations can have duplicate elements.

Arguments

- **iterable** (`Iterable()`) – The iterable to combine over.
- **r** (`number()`) – The number of elements to combine.

Returns **Generator** – - A generator that yields the combinations.

compress(*data*, *selectors*)

Return elements from an iterable as long as the predicate is true.

Arguments

- **data** (`Iterable()`) – The iterable to filter.
- **selectors** (`Iterable()`) – The iterable of booleans to filter by.

Returns **Generator** – - A generator that yields the filtered elements.

count(*start=0*, *step=1*)

Return an infinite iterator of integers starting at `start` (or 0 if not provided), incremented by `step` (or 1 if not provided). `step` defaults to 1.

Arguments

- **start** – The starting value (default 0).

- **step** – The increment step (default 1).

Returns An iterator of integers.

cycle(*iterable*)

Return an iterator that returns elements from the iterable and then repeats the same sequence forever.

Arguments

- **iterable** (`Iterable()`) – The iterable to repeat.

Returns **Generator** – - A generator that yields the repeated elements.

dropwhile(*predicate*, *iterable*)

Return an iterator that drops elements from the iterable as long as the predicate is true.

Arguments

- **predicate** (`function()`) – The function to filter by.
- **iterable** (`Iterable()`) – The iterable to filter.

filterfalse(*predicate*, *iterable=null*)

Return an iterator that filters elements from the iterable as long as the predicate is true.

Arguments

- **predicate** (`function()`) – The function to filter by.
- **iterable** (`Iterable()`) – The iterable to filter.

groupby(*iterable*, *key=null*)

Return an iterator that groups elements from the iterable into a sequence of tuples. The first element of each tuple is the key to the group. The second element of each tuple is the iterator of the group.

Arguments

- **iterable** (`Iterable()`) – The iterable to group.
- **key** (`function()`) – The function to group by.

Returns **Generator** – - A generator that yields the grouped elements.

islice(*iterable*, *start=0*, *stop*, *step=1*)

Return an iterator that returns selected elements from the iterable.

Arguments

- **iterable** (`Iterable()`) – The iterable to slice.
- **start** (`number()`) – The starting index.
- **stop** (`number()`) – The ending index.
- **step** (`number()`) – The step size.

Returns **Generator** – - A generator that yields the sliced elements.

pairwise(*iterable*)

Return an iterator that returns pairs of elements from the iterable.

Arguments

- **iterable** (`Iterable()`) – The iterable to pair.

Returns **Generator** – - A generator that yields the paired elements.

permutations(*iterable*, *r=null*)

Return an iterator that returns permutations of the elements from the iterable.

Arguments

- **iterable** (Iterable()) – The iterable to permute.
- **r** (number()) – The number of elements to permute.

Returns **Generator** – - A generator that yields the permuted elements.

product(*repeat=1*, *...args*)

Return an iterator that returns the cartesian product of the iterables.

Arguments

- **repeat** (number()) – The number of times to repeat the product.
- **args** (*any()*) – The iterables to product.

Returns **Generator** – - A generator that yields the cartesian product.

repeat(*object*, *times=null*)

Return an iterator that returns the same element n times.

Arguments

- **object** (*any()*) – The element to repeat.
- **times** (number()) – The number of times to repeat the element.

starmap(*func*, *iterable*)

Return an iterator that returns the results of applying func to the elements from the iterable.

Arguments

- **func** (function()) – The function to apply to the elements.
- **iterable** (Iterable()) – The iterable to apply the function to.

Returns **Generator** – - A generator that yields the results of applying func to the elements.

takewhile(*predicate*, *iterable*)

Return an iterator that returns elements from the iterable as long as the predicate is true.

Arguments

- **predicate** (function()) – The predicate function.
- **iterable** (Iterable()) – The iterable to take elements from.

Returns **Generator** – - A generator that yields the elements from the iterable.

tee(*iterable*, *n=2*)

Return an iterator that returns n independent iterators of the iterable.

Arguments

- **iterable** (Iterable()) – The iterable to tee.
- **n** (number()) – The number of iterators to return.

Returns **Generator** – - A generator that yields the iterators.

zip_longest(*fillvalue=null*, *...args*)

Return an iterator that returns elements from the iterables. The iterator stops when the longest iterable is exhausted. If the iterables are of different lengths, missing values are filled with the fillvalue.

Arguments

- **fillvalue** (Any()) – The fillvalue to use.
- **args** (*any()*) – The iterables to zip.

Returns **Generator** – - A generator that yields the zipped elements.

1.6 String

1.6.1 Constants

ascii_letters

The concatenation of the ascii_lowercase and ascii_uppercase constants

ascii_lowercase

The lowercase letters 'abcdefghijklmnopqrstuvwxyz'.

ascii_uppercase

The uppercase letters 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.

digits

The string '0123456789'.

hexdigits

The string '0123456789abcdefABCDEF'.

octdigits

The string '01234567'.

printable

String of ASCII characters which are considered printable. This is a combination of the ascii_letters, digits, punctuation and whitespace constants.

punctuation

String of ASCII characters which are considered punctuation characters in the C locale. !"#$%&'()*+,-./:;<=>?@[]^_`{|}~

whitespace

The whitespace characters ' \t\n\r\v\f'.

1.7 Indices and tables

- genindex

INDEX

T

U

W

Z