Appendix N. Converting DOS Batch Files to Shell Scripts

Quite a number of programmers learned scripting on a PC running DOS. Even the crippled DOS batch file language allowed writing some fairly powerful scripts and applications, though they often required extensive kludges and workarounds. Occasionally, the need still arises to convert an old DOS batch file to a UNIX shell script. This is generally not difficult, as DOS batch file operators are only a limited subset of the equivalent shell scripting ones.

Table N-1. Batch file keywords / variables / operators, and their shell equivalents

Batch File Operator	Shell Script Equivalent	Meaning
%	$	command-line parameter prefix
/	-	command option flag
\	/	directory path separator
==	=	(equal-to) string comparison test
!==!	!=	(not equal-to) string comparison test
|	|	pipe
@	set +v	do not echo current command
*	*	filename "wild card"
>	>	file redirection (overwrite)
>>	>>	file redirection (append)
<	<	redirect stdin
%VAR%	$VAR	environmental variable
REM	#	comment
NOT	!	negate following test
NUL	/dev/null	"black hole" for burying command output
ECHO	echo	echo (many more option in Bash)
ECHO.	echo	echo blank line
ECHO OFF	set +v	do not echo command(s) following
FOR %%VAR IN (LIST) DO	for var in [list]; do	"for" loop
:LABEL	none (unnecessary)	label
GOTO	none (use a function)	jump to another location in the script
PAUSE	sleep	pause or wait an interval
CHOICE	case or select	menu choice
IF	if	if-test
IF EXIST *FILENAME*	if [-e filename]	test if file exists
IF !%N==!	if [-z "$N"]	if replaceable parameter "N" not present
CALL	source or . (dot operator)	"include" another script
COMMAND /C	source or . (dot operator)	"include" another script (same as CALL)
SET	export	set an environmental variable
SHIFT	shift	left shift command-line argument list
SGN	-lt or -gt	sign (of integer)
ERRORLEVEL	$?	exit status
CON	stdin	"console" (stdin)
PRN	/dev/lp0	(generic) printer device
LPT1	/dev/lp0	first printer device
COM1	/dev/ttyS0	first serial port

Batch files usually contain DOS commands. These must be translated into their UNIX equivalents in order to convert a batch file into a shell script.

Table N-2. DOS commands and their UNIX equivalents

DOS Command	UNIX Equivalent	Effect
ASSIGN	ln	link file or directory
ATTRIB	chmod	change file permissions
CD	cd	change directory

DOS Command	UNIX Equivalent	Effect
CHDIR	cd	change directory
CLS	clear	clear screen
COMP	diff, comm, cmp	file compare
COPY	cp	file copy
Ctl-C	Ctl-C	break (signal)
Ctl-Z	Ctl-D	EOF (end-of-file)
DEL	rm	delete file(s)
DELTREE	rm -rf	delete directory recursively
DIR	ls -l	directory listing
ERASE	rm	delete file(s)
EXIT	exit	exit current process
FC	comm, cmp	file compare
FIND	grep	find strings in files
MD	mkdir	make directory
MKDIR	mkdir	make directory
MORE	more	text file paging filter
MOVE	mv	move
PATH	$PATH	path to executables
REN	mv	rename (move)
RENAME	mv	rename (move)
RD	rmdir	remove directory
RMDIR	rmdir	remove directory
SORT	sort	sort file
TIME	date	display system time
TYPE	cat	output file to stdout
XCOPY	cp	(extended) file copy

Virtually all UNIX and shell operators and commands have many more options and enhancements than their DOS and batch file counterparts. Many DOS batch files rely on auxiliary utilities, such as **ask.com**, a crippled counterpart to read.

DOS supports only a very limited and incompatible subset of filename wild-card expansion, recognizing just the * and ? characters.

Converting a DOS batch file into a shell script is generally straightforward, and the result ofttimes reads better than the original.

Example N-1. VIEWDATA.BAT: DOS Batch File

```
REM VIEWDATA

REM INSPIRED BY AN EXAMPLE IN "DOS POWERTOOLS"
REM                           BY PAUL SOMERSON


@ECHO OFF

IF !%1==! GOTO VIEWDATA
REM  IF NO COMMAND-LINE ARG...
FIND "%1" C:\BOZO\BOOKLIST.TXT
GOTO EXIT0
REM  PRINT LINE WITH STRING MATCH, THEN EXIT.

:VIEWDATA
TYPE C:\BOZO\BOOKLIST.TXT | MORE
REM  SHOW ENTIRE FILE, 1 PAGE AT A TIME.

:EXIT0
```

The script conversion is somewhat of an improvement. [1]

Example N-2. *viewdata.sh*: Shell Script Conversion of VIEWDATA.BAT

```
#!/bin/bash
# viewdata.sh
# Conversion of VIEWDATA.BAT to shell script.

DATAFILE=/home/bozo/datafiles/book-collection.data
ARGNO=1

# @ECHO OFF                 Command unnecessary here.

if [ $# -lt "$ARGNO" ]    # IF !%1==! GOTO VIEWDATA
then
  less $DATAFILE          # TYPE C:\MYDIR\BOOKLIST.TXT | MORE
else
  grep "$1" $DATAFILE     # FIND "%1" C:\MYDIR\BOOKLIST.TXT
fi

exit 0                    # :EXIT0

#  GOTOs, labels, smoke-and-mirrors, and flimflam unnecessary.
#  The converted script is short, sweet, and clean,
#+ which is more than can be said for the original.
```

Ted Davis' Shell Scripts on the PC site had a set of comprehensive tutorials on the old-fashioned art of batch file programming. Unfortunately the page has vanished without a trace.

Notes

[1] Various readers have suggested modifications of the above batch file to prettify it and make it more compact and efficient. In the opinion of the *ABS Guide* author, this is wasted effort. A Bash script can access a DOS filesystem, or even an NTFS partition (with the help of ntfs-3g) to do batch or scripted operations.

Are you writing print() statements to debug your Python code?

Pradeepa Gollapalli
Jul 21 · 7 min read ★

Then, you should read this...

Photo by Geoff Park on Unsplash

I was one such person who used to debug code using print() statements. Some times, if the code is lengthy, then there are more prints with multiple symbols to differentiate from one another.

Have a look at the code snippet below.
(Code snippets in this blog, follow the syntax of Python 3.7)

```
print(style_dict,"
{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}}]]]]]]]]]
]]]]]]]]]]]")
# Adding into a dictionary
res_dct = {style_dict[i]: style_dict[i + 1] for i in range(0,
len(style_dict), 2)}
res_dist={res_dct['Email Address']:{style_dict[i]: style_dict[i + 1]
for i in range(0, len(style_dict), 2)}}
print(res_dist,"++++++++++++++++++++++++++++++++++++++++++++++++++++
+")
recon_dict = res_dct
print(recon_dict,"--------------------------------------------------
-")
# Removing space so that data can be transferred to HTML fields
recon_dict = {x.translate({32: None}): y
              for x, y in list(recon_dict.items())}
print("##################################################")
print(recon_dict)
print("################################################")

# Converting to JSON
r = json.dumps(recon_dict)
print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$")
print(r)
loaded_json = json.loads(r)
print("WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWwwwwww")
print("******************************************************")
print(loaded_json)
```

Here, I’m trying to add a dictionary to a JSON file. Due to some error, I had to use that many print statements with different symbols to debug.

But as the code gets bigger with different modules and different classes calling different definitions in other modules or classes, this is not the right choice.

Let’s see some drawbacks for this approach:

- With more code being added, it's difficult to use print statements in every module, class, or definition the code is traversing through.
- Even before we notice the bug, the code gets executed and goes to the next steps.
- Wait till a lengthy execution is complete to find and make a fix.

- Going back to a whole lot of logs to search the right symbols we gave in the print statements and matching them is tedious.

We felt giving prints is a simple debugging solution but, doesn't it feel tedious now.

A Simple turnaround,

We don't have to do anything except using a powerful weapon that Python has provides us, the "pdb module". This module helps us to debug effectively.

What is pdb(python debugger)?
pdb is an interactive shell, that helps to debug python code. It helps us to step into our code, one line at a time, pause, examine the state, and continue to next line of code or continue execution.

Some ways to invoke pdb:

Here, we are looking at 3 ways to invoke pdb.
Postmortem: Use this, if you want to debug at the program level.
Inline pdb: Use this, if you are working on versions earlier to 3.7
breakpoint(): Use this, for version 3.7 and higher

a) Postmortem

Let's understand with a simple program.

debug_add.py

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Here the def add_num is supposed to add the value of num variable to each element in a list named listA, store new values in list sum, and return list sum.

By executing the python file as below will invoke pdb,

```
python -m pdb debug_add.py
```

This will enter the pdb mode and halt at the first line of code.

```
(venv) C:\Users\PycharmProjects\>python -m pdb debug_add.py
> c:\users\pycharmprojects\debug_add.py(2)<module>()
-> def add_num(listA,num):
(Pdb)
```

Anytime if you need help with debugger use ‘h’(help), which lists all the options.

```
(Pdb) h

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv
undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source
until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u
whatis
bt     continue   exit     l         pp        run      unalias
where

Miscellaneous help topics:
==========================
exec  pdb
```

Help on a specific option,

```
(Pdb) h debug
debug code
        Enter a recursive debugger that steps through the code
        argument (which is an arbitrary expression or statement to
        be executed in the current environment).
```

Back to the program, to go to the next step of execution use option ‘n’(next).

```
> c:\users\pycharmprojects\debug_add.py(2)<module>()
-> def add_num(listA,num):
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(8)<module>()
-> listA = [2, 4, 6, 8]
```

Here we can examine the values of a variable by giving the name of the variable as below,

```
(Pdb) listA
*** NameError: name 'listA' is not defined
(Pdb)
*** NameError: name 'listA' is not defined
```

We reached the line listA = [2, 4, 6, 8] but we still haven't executed, so it says listA not defined. If you observed if we hit enter at any time the previous option gets executed as above.

Now hit 'n' to move forward and check the listA variable.

```
(Pdb) n
> c:\users\pycharmprojects\debug_add.py(9)<module>()
-> num=10
(Pdb) listA
[2, 4, 6, 8]
(Pdb)
```

To check which line of code we are in, use the option 'l'(line). The arrow mark points to the line we are in, EOF represents end of file.

```
(Pdb) l
  4         for i in listA:
  5             sum.append(i*num)
  6         return sum
  7
  8     listA = [2, 4, 6, 8]
  9  -> num=10
 10     result=add_num(listA,num)
 11     print(result)
[EOF]
(Pdb)
```

To quit debugger, we use option ‘q’(quit).

```
(Pdb) q

(venv) C:\Users\PycharmProjects\>
```

Another way to go with postmortem method is to halt the execution, only when an exception is encountered, For this use -c continue along with -m pdb.

```
python -m pdb -c continue debug_add.py
```

b) Inline pdb

In earlier versions of Python 3.7, we have to explicitly import the module pdb and call pdb.set_trace() to halt the program and perform debugging.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
import pdb; pdb.set_trace()
result=add_num(listA,num)
print(result)
```

Let’s examine the console when we run this.

```
> c:\users\pycharmprojects\debug_add.py(11)<module>()
-> result=add_num(listA,num)
(Pdb)
```

The execution halted, as pdb.set_trace() is encountered and entered debug mode. We can now perform all the debugging actions here as shown in the postmortem method above.

c) breakpoint()

From Python 3.7 onwards, breakpoint() definition is introduced which helps to debug pythonic code without having to explicitly import the module pdb and call pdb.set_trace(). breakpoint() does all that for us and opens PDB debugger in the console.

Now, let's execute the above code without any breakpoints and debug if any errors are encountered.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Output:

```
C:\Users\PycharmProjects\venv\Scripts\python.exe
C:/Users/PycharmProjects/debug_add.py
[20, 40, 60, 80]

Process finished with exit code 0
```

The mission of the code block is, to add num which is 10 to each of the elements in the list and return the new list.
Expected result is [12, 14, 16, 18]
Actual result is [20, 40, 60, 80]

Now let's use the breakpoint() weapon to debug and fix the code.

Where you place breakpoint() depends on where you suspect the bug is. In this case, we place it before it enters the add_num() definition.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
breakpoint()
result=add_num(listA,num)
print(result)
```

Output:

```
> c:\users\pycharmprojects\debug_add.py(11)<module>()
-> result=add_num(listA,num)
(Pdb) n
> c:\users\pycharmprojects\debug_add.py(12)<module>()
-> print(result)
(Pdb) n
[20, 40, 60, 80]
 — Return —
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(12)<module>
()->None
-> print(result)
(Pdb)
```

Option ’n’(next) is used to move to the next line or step over any definitions. But in this case, we need to step into the definition, for that we will use option ‘s’(step).

Below the text in bold is used to highlight the option used and its explanation.

```
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(11)<module>
()
-> result=add_num(listA,num)
(Pdb) s  <----- Step into def add_num
--Call--
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(2)add_num()
-> def add_num(listA,num):
(Pdb) s  <---- stepped inside def add_num
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(3)add_num()
-> sum=[]
(Pdb) n  <--- inside a def feel free to use 'n'
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(4)add_num()
-> for i in listA:
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(5)add_num()
```

```
-> sum.append(i*num)
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(4)add_num()
-> for i in listA:
(Pdb) sum  <-- examine sum value
[20] <--- 2+10 =12 not 20,oops we used '*'instead of '+' in
                          appending to list sum,CAUGHT IT!
(Pdb) i  <-- so, examine i
2
(Pdb) sum.append(i+num) <-- try adding + in the expression
(Pdb) sum
[20, 12] <-- PERFECT, FIXED IT!
(Pdb) u <-- used to skip other iterations of for loop.
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(11)<module>
()
-> result=add_num(listA,num)
(Pdb) c <-- used to continue with execution
[20, 12, 40, 60, 80] <--not a right answer but found a fix.

Process finished with exit code 0
```

Above, after the first iteration of the for loop, we examined the sum value and it displayed 20 instead of 12. We almost caught it here that we misplaced * (multiplication) in place of + (addition). So then, we took a step ahead to examined 'i' which is 2 at that point and tried **sum.append(i+num).** Then examining sum gave us 12 as recently added element. Hence we got the fix, so we skipped the remaining iterations of for loop by using the option 'u' (until). Then it moved to next step after loop. Here we used 'c'(continue) to continue the execution and it ended.

Now the fix,

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i+num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Output:

```
C:\Users\PycharmProjects\venv\Scripts\python.exe
C:/Users/PycharmProjects/debug_add.py
```

```
[12, 14, 16, 18]

Process finished with exit code 0
```

Hence debugged!!

Doesn't it seem easy, with no mess of print() statements?

Little recap to the options of pdb used here:

> *n — move to next line/step over definitions*
> *s — step into definitions (built-in / user defined)*
> *u — to skip remaining iterations in a loop*
> *c — continue execution or till the next breakpoint() is encountered*
> *l — Shows the current line of code to be executed with arrow "->"*
> *q — to quit the debugger*
> *variable name — to examine the current state of the variable*
> *h option name — displays help on the option provided*
> *h — to view options menu and explore more options as needed.*

Conclusion

pdb is a powerful weapon to debug Pythonic code which adds "effectiveness" as there's no mess of print()statements in your code and "efficiency" as it greatly reduces the time to debug.

With a bunch of options provided, it should be your "goto" tool when you encounter your next bug.

Happy Debugging!!

Source

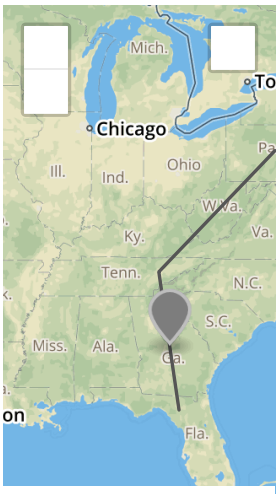
Python3 Python Debugger Python Programming Debugging Good Coding Practice


Download on the
App Store


GET IT ON
Google Play

along

Takes a LineString and returns a Point at a specified distance along the line.



© Mapbox © OpenStreetMap

npm install @turf/along

Arguments

Argument	Type	Description
line	Feature <LineString>	input line
distance	number	distance along the line
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	can be degrees, radians, miles, or kilometers

Returns

Feature <Point> - Point distance units along the line

Example

```
var line = turf.lineString([[-83, 30], [-84, 36], [-78, 41]]
var options = {units: 'miles'};

var along = turf.along(line, 200, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

area

npm install @turf/area

Takes one or more features and returns their area in square meters.

Arguments

Argument	Type	Description
geojson	GeoJSON	input GeoJSON feature(s)

Returns

number - area in square meters

Example

```
var polygon = turf.polygon([[[125, -15], [113, -22], [154, -
var area = turf.area(polygon);
```

bbox

npm install @turf/bbox

Takes a set of features, calculates the bbox of all input features, and returns a bounding box.

Arguments

Argument	Type	Description
geojson	GeoJSON	any GeoJSON object

Returns

BBox - bbox extent in minX, minY, maxX, maxY order

Example

```
var line = turf.lineString([[-74, 40], [-78, 42], [-82, 35]]
var bbox = turf.bbox(line);
```

```
var bboxPolygon = turf.bboxPolygon(bbox);
```


bboxPolygon

npm install @turf/bbox-polygon

Takes a bbox and returns an equivalent polygon.

Arguments

Argument	Type	Description
bbox	BBox	extent in minX, minY, maxX, maxY order
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Properties	{}	Translate properties to Polygon
id	(string|number)	{}	Translate Id to Polygon

Returns

Feature <Polygon> - a Polygon representation of the bounding box

Example

```
var bbox = [0, 0, 10, 10];

var poly = turf.bboxPolygon(bbox);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

bearing

npm install @turf/bearing

Takes two points and finds the geographic bearing between them, i.e. the angle measured in degrees from the north line (0 degrees)

Arguments

Argument	Type	Description
start	Coord	starting Point
end	Coord	ending Point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
final	boolean	false	calculates the final bearing if true

Returns

number - bearing in decimal degrees, between -180 and 180 degrees (positive clockwise)

Example

```
var point1 = turf.point([-75.343, 39.984]);
var point2 = turf.point([-75.534, 39.123]);

var bearing = turf.bearing(point1, point2);
```

center

npm install @turf/center

Takes a Feature or FeatureCollection and returns the absolute center point of all features.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	an Object that is used as the

Returns

Feature <Point> - a Point feature at the absolute center point of all input features

Example

```
var features = turf.featureCollection([
  turf.point( [-97.522259, 35.4691]),
  turf.point( [-97.502754, 35.463455]),
  turf.point( [-97.508269, 35.463245])
]);

var center = turf.center(features);
```

centerOfMass

npm install @turf/center-of-mass

Takes any Feature or a FeatureCollection and returns its center of mass using this formula: Centroid of Polygon.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
properties	Object	an Object that is used as the Feature 's properties

Returns

Feature <Point> - the center of mass

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31],

var center = turf.centerOfMass(polygon);
```

centroid

npm install @turf/centroid

Takes one or more features and calculates the centroid using the mean of all vertices. This lessens the effect of small islands and artifacts when calculating the centroid of a set of polygons.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered
properties	Object	an Object that is used as the Feature 's properties

Returns

Feature <Point> - the centroid of the input features

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31],

var centroid = turf.centroid(polygon);
```

destination

npm install @turf/destination

Takes a Point and calculates the location of a destination point given a distance in degrees, radians, miles, or kilometers; and bearing in degrees. This uses the Haversine formula to account for global curvature.

Arguments

Argument	Type	Description
origin	Coord	starting point
distance	number	distance from the origin point
bearing	number	ranging from -180 to 180
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	miles, kilometers, degrees, or radians
properties	Object	{}	Translate properties to Point

Returns

Feature <Point> - destination point

Example

```
var point = turf.point([-75.343, 39.984]);
var distance = 50;
var bearing = 90;
var options = {units: 'miles'};
```

```
var destination = turf.destination(point, distance, bearing,
```


distance

npm install @turf/distance

Calculates the distance between two points in degrees, radians, miles, or kilometers. This uses the Haversine formula to account for global curvature.

Arguments

Argument	Type	Description
from	Coord	origin point
to	Coord	destination point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

number - distance between the two points

Example

```
var from = turf.point([-75.343, 39.984]);
var to = turf.point([-75.534, 39.123]);
var options = {units: 'miles'};

var distance = turf.distance(from, to, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

envelope

npm install @turf/envelope

Takes any number of features and returns a rectangular Polygon that encompasses all vertices.

Arguments

Argument	Type	Description
geojson	GeoJSON	input features

Returns

Feature <Polygon> - a rectangular Polygon feature that encompasses all vertices

Example

```
var features = turf.featureCollection([
  turf.point([-75.343, 39.984], {"name": "Location A"}),
  turf.point([-75.833, 39.284], {"name": "Location B"}),
  turf.point([-75.534, 39.123], {"name": "Location C"})
]);

var enveloped = turf.envelope(features);
```

length

npm install @turf/length

Takes a GeoJSON and measures its length in the specified units, (Multi)Point 's distance are ignored.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to measure
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	kilometers	can be degrees, radians, miles, or kilometers

Returns

number - length of GeoJSON

Example

```
var line = turf.lineString([[115, -32], [131, -22], [143, -2
var length = turf.length(line, {units: 'miles'});
```

midpoint

npm install @turf/midpoint

Takes two points and returns a point midway between them. The midpoint is calculated geodesically, meaning the curvature of the earth is taken into account.

Arguments

Argument	Type	Description
point1	Coord	first point
point2	Coord	second point

Returns

Feature <Point> - a point midway between pt1 and pt2

Example

```
var point1 = turf.point([144.834823, -37.771257]);
var point2 = turf.point([145.14244, -37.830937]);

var midpoint = turf.midpoint(point1, point2);
```

pointOnFeature

npm install @turf/point-on-feature

Takes a Feature or FeatureCollection and returns a Point guaranteed to be on the surface of the feature.

Arguments

Argument	Type	Description
geojson	GeoJSON	any Feature or FeatureCollection

Returns

Feature <Point> - a point on the surface of input

Example

```
var polygon = turf.polygon([[
  [116, -36],
  [131, -32],
  [146, -43],
  [155, -25],
  [133, -9],
  [111, -22],
  [116, -36]
]]);

var pointOnPolygon = turf.pointOnFeature(polygon);
```

polygonTangent

npm install @turf/polygon-tangents

Finds the tangents of a (Multi)Polygon from a Point.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Arguments

Argument	Type	Description
pt	Coord	to calculate the tangent points from
polygon	Feature <(Polygon|MultiPolygon)>	to get tangents from

Returns

FeatureCollection <Point> - Feature Collection containing the two tangent points

Example

```
var polygon = turf.polygon([[[11, 0], [22, 4], [31, 0], [31,
var point = turf.point([61, 5]);

var tangents = turf.polygonTangents(point, polygon)
```

pointToLineDista

npm install @turf/point-to-line-distance

Returns the minimum distance between a Point and a LineString , being the distance from a line the minimum distance between the point and any segment of the LineString.

Arguments

Argument	Type	Description
pt	Coord	Feature or Geometry
line	Feature <LineString>	GeoJSON Feature or Geometry
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers
mercator	boolean	false	if distance should be on Mercator or WGS84 projection

Returns

number - distance between point and line

Example

```
var pt = turf.point([0, 0]);
var line = turf.lineString([[1, 1],[-1, 1]]);

var distance = turf.pointToLineDistance(pt, line, {units: 'm
//=69.11854715938406
```

rhumbBearing

npm install @turf/rhumb-bearing

Takes two points and finds the bearing angle between them along a Rhumb line i.e. the angle measured in degrees start the north line (0 degrees)

Arguments

Argument	Type	Description
start	Coord	starting Point
end	Coord	ending Point
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
final	boolean	false	calculates the final bearing if true

Returns

number - bearing from north in decimal degrees, between -180 and 180 degrees (positive clockwise)

Example

```
var point1 = turf.point([-75.343, 39.984], {"marker-color": '
var point2 = turf.point([-75.534, 39.123], {"marker-color": '

var bearing = turf.rhumbBearing(point1, point2);
```

rhumbDestinatic

npm install @turf/rhumb-destination

Returns the destination Point having travelled the given distance along a Rhumb line from the origin Point with the (varant) given bearing.

Arguments

Argument	Type	Description
origin	Coord	starting point
distance	number	distance from the starting point
bearing	number	varant bearing angle ranging from -180 to 180 degrees from north
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers
properties	Object	{}	translate properties to destination point

Returns

Feature <Point> - Destination point.

Example

```
var pt = turf.point([-75.343, 39.984], {"marker-color": "F00'
var distance = 50;
var bearing = 90;
var options = {units: 'miles'};

var destination = turf.rhumbDestination(pt, distance, bearing
```

rhumbDistance

npm install @turf/rhumb-distance

Calculates the distance along a rhumb line between two points in degrees, radians, miles, or kilometers.

Arguments

Argument	Type	Description
from	Coord	origin point
to	Coord	destination point
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	can be degrees, radians, miles, or kilometers

Returns

number - distance between the two points

Example

```
var from = turf.point([-75.343, 39.984]);
var to = turf.point([-75.534, 39.123]);
var options = {units: 'miles'};

var distance = turf.rhumbDistance(from, to, options);
```

square

npm install @turf/square

Takes a bounding box and calculates the minimum square bounding box that would contain the input.

Arguments

Argument	Type	Description
bbox	BBox	extent in west, south, east, north order

Returns

BBox - a square surrounding bbox

Example

```
var bbox = [-20, -20, -15, 0];
var squared = turf.square(bbox);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

greatCircle

npm install @turf/great-circle

Calculate great circles routes as LineString

Arguments

Argument	Type	Description
start	Coord	source point feature
end	Coord	destination point feature
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	line feature properties
npoints	number	100	number of points
offset	number	10	offset controls the likelyhood that lines will be split which cross the dateline. The higher the number the more likely.

Returns

Feature <LineString> - great circle line feature

Example

```
var start = turf.point([-122, 48]);
var end = turf.point([-77, 39]);

var greatCircle = turf.greatCircle(start, end, {'name': 'Sea
```

cleanCoords

npm install @turf/clean-coords

Removes redundant coordinates from any GeoJSON Geometry.

Arguments

Argument	Type	Description
geojson	(Geometry|Feature)	Feature or Geometry
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated

Returns

(Geometry|Feature) - the cleaned input Feature/Geometry

Example

```
var line = turf.lineString([[0, 0], [0, 2], [0, 5], [0, 8],
var multiPoint = turf.multiPoint([[0, 0], [0, 0], [2, 2]]);

turf.cleanCoords(line).geometry.coordinates;
//= [[0, 0], [0, 10]]

turf.cleanCoords(multiPoint).geometry.coordinates;
//= [[0, 0], [2, 2]]
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

flip

npm install @turf/flip

Takes input features and flips all of their coordinates from [x, y] to [y, x].

Arguments

Argument	Type	Description
geojson	GeoJSON	input features
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - a feature or set of features of the same type as input with flipped coordinates

Example

```
var serbia = turf.point([20.566406, 43.421008]);

var saudiArabia = turf.flip(serbia);
```

rewind

npm install @turf/rewind

Rewind (Multi)LineString or (Multi)Polygon outer ring counterclockwise and inner rings clockwise (Uses Shoelace Formula).

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Arguments

Argument	Type	Description
geojson	GeoJSON	input GeoJSON Polygon
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
reverse	boolean	false	enable reverse winding
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - rewind Polygon

Example

```
var polygon = turf.polygon([[[121, -29], [138, -29], [138, -1

var rewind = turf.rewind(polygon);
```

round

npm install @turf/helpers

Round number to precision

Note: round is part of the @turf/helpers module.

To use it as a

Arguments

Argument	Type	Description
num	number	Number

Argument	Type	Description
precision	number	Precision

stand-alone module will need to import @turf/helpers and call the round method.

Returns

number - rounded number

Example

```
turf.round(120.4321)
//=120

turf.round(120.4321, 2)
//=120.43
```

truncate

npm install @turf/truncate

Takes a GeoJSON Feature or FeatureCollection and truncates the precision of the geometry.

Arguments

Argument	Type	Description
geojson	GeoJSON	any GeoJSON Feature, FeatureCollection, Geometry or GeometryCollection.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
precision	number	6	coordinate decimal precision

Prop	Type	Default	Description
coordinates	number	3	maximum number of coordinates (primarly used to remove z coordinates)
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - layer with truncated geometry

Example

```
var point = turf.point([
    70.46923055566859,
    58.11088890802906,
    1508
]);
var options = {precision: 3, coordinates: 2};
var truncated = turf.truncate(point, options);
//=truncated.geometry.coordinates => [70.469, 58.111]
```

bboxClip

npm install @turf/bbox-clip

Takes a Feature and a bbox and clips the feature to the bbox using lineclip. May result in degenerate edges when clipping Polygons.

Arguments

Argument	Type	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
feature	Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>	feature to clip to the bbox
bbox	BBox	extent in minX, minY, maxX, maxY order

Returns

Feature <(LineString|MultiLineString|Polygon|MultiPolygon)> - clipped Feature

Example

```
var bbox = [0, 0, 10, 10];
var poly = turf.polygon([[[2, 2], [8, 4], [12, 8], [3, 7], [
var clipped = turf.bboxClip(poly, bbox);
```

bezierSpline

npm install @turf/bezier-spline

Takes a line and returns a curved version by applying a Bezier spline algorithm.

Arguments

Argument	Type	Description
line	Feature <LineString>	input LineString
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
resolution	number	10000	time in milliseconds between points
sharpness	number	0.85	a measure of how curvy the path should be between splines

Returns

Feature <LineString> - curved line

Example

```
var line = turf.lineString([
  [-76.091308, 18.427501],
  [-76.695556, 18.729501],
  [-76.552734, 19.40443],
  [-74.61914, 19.134789],
  [-73.652343, 20.07657],
  [-73.157958, 20.210656]
]);

var curved = turf.bezierSpline(line);
```

buffer

npm install @turf/buffer

Calculates a buffer for input features for a given radius. Units supported are miles, kilometers, and degrees.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Geometry|Feature)	input to be buffered

Argument	Type	Description
radius	number	distance to draw the buffer (negative values are allowed)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	any of the options supported by turf units
steps	number	64	number of steps

Returns

(FeatureCollection|Feature <(Polygon|MultiPolygon)>|undefined) - buffered features

Example

```
var point = turf.point([-90.548630, 14.616599]);
var buffered = turf.buffer(point, 500, {units: 'miles'});
```

circle

npm install @turf/circle

Takes a Point and calculates the circle polygon given a radius in degrees, radians, miles, or kilometers; and steps for precision.

Arguments

Argument	Type	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
center	(Feature <Point>|Array)	center point
radius	number	radius of the circle
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
steps	number	64	number of steps
units	string	'kilometers'	miles, kilometers, degrees, or radians
properties	Object	{}	properties

Returns

Feature <Polygon> - circle polygon

Example

```
var center = [-75.343, 39.984];
var radius = 5;
var options = {steps: 10, units: 'kilometers', properties: {
var circle = turf.circle(center, radius, options);
```

clone

npm install @turf/clone

Returns a cloned copy of the passed GeoJSON Object, including possible 'Foreign Members'. ~3-5x faster than the common JSON.parse + JSON.stringify combo method.

Arguments

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
geojson	GeoJSON	GeoJSON Object

Returns

GeoJSON - cloned GeoJSON Object

Example

```
var line = turf.lineString([[-74, 40], [-78, 42], [-82, 35]]

var lineCloned = turf.clone(line);
```

concave

npm install @turf/concave

Takes a set of points and returns a concave hull Polygon or MultiPolygon. Internally, this uses turf-tin to generate geometries.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	input points
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
maxEdge	number	Infinity	the length (in 'units') of an edge necessary for part of the hull to become concave.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

(Feature <(Polygon|MultiPolygon)>|null) - a concave hull (null value is returned if unable to compute hull)

Example

```
var points = turf.featureCollection([
  turf.point([-63.601226, 44.642643]),
  turf.point([-63.591442, 44.651436]),
  turf.point([-63.580799, 44.648749]),
  turf.point([-63.573589, 44.641788]),
  turf.point([-63.587665, 44.64533]),
  turf.point([-63.595218, 44.64765])
]);
var options = {units: 'miles', maxEdge: 1};

var hull = turf.concave(points, options);
```

convex

npm install @turf/convex

Takes a Feature or a FeatureCollection and returns a convex hull Polygon.

Arguments

Argument	Type	Description
geojson	GeoJSON	input Feature or FeatureCollection
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
concavity	number	Infinity	1 - thin shape. Infinity - convex hull.

Returns

Feature <Polygon> - a convex hull

Example

```
var points = turf.featureCollection([
  turf.point([10.195312, 43.755225]),
  turf.point([10.404052, 43.8424511]),
  turf.point([10.579833, 43.659924]),
  turf.point([10.360107, 43.516688]),
  turf.point([10.14038, 43.588348]),
  turf.point([10.195312, 43.755225])
]);

var hull = turf.convex(points);
```

difference

npm install @turf/difference

Finds the difference between two polygons by clipping the second polygon from the first.

Arguments

Argument	Type	Description
polygon1	Feature <(Polygon|MultiPolygon)>	input Polygon feature
polygon2	Feature <(Polygon|MultiPolygon)>	Polygon feature to difference from polygon1

Returns

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

(Feature <(Polygon|MultiPolygon)>|null) - a Polygon or MultiPolygon feature showing the area of polygon1 excluding the area of polygon2 (if empty returns null)

Example

```
var polygon1 = turf.polygon([[
  [128, -26],
  [141, -26],
  [141, -21],
  [128, -21],
  [128, -26]
]], {
  "fill": "#F00",
  "fill-opacity": 0.1
});
var polygon2 = turf.polygon([[
  [126, -28],
  [140, -28],
  [140, -20],
  [126, -20],
  [126, -28]
]], {
  "fill": "#00F",
  "fill-opacity": 0.1
});

var difference = turf.difference(polygon1, polygon2);
```

dissolve

npm install @turf/dissolve

Dissolves a FeatureCollection of polygon features, filtered by an optional property name:value. Note that mulitpolygon features within the collection are not supported

Arguments

Argument	Type	Description

Argument	Type	Description
featureCollection	FeatureCollection <Polygon>	input feature collection to be dissolved
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
propertyName	(string)		features with equals 'propertyName' in

Returns

FeatureCollection <Polygon> - a FeatureCollection containing the dissolved polygons

Example

```
var features = turf.featureCollection([
  turf.polygon([[[0, 0], [0, 1], [1, 1], [1, 0], [0, 0]]], {
  turf.polygon([[[0, -1], [0, 0], [1, 0], [1, -1], [0,-1]]],
  turf.polygon([[[1,-1],[1, 0], [2, 0], [2, -1], [1, -1]]],
]);

var dissolved = turf.dissolve(features, {propertyName: 'comb
```

intersect

npm install @turf/intersect

Takes two polygons and finds their intersection. If they share a border, returns the border; if they don't intersect, returns undefined.

Arguments

Argument	Type	Description

Argument	Type	Description
poly1	Feature <Polygon>	the first polygon
poly2	Feature <Polygon>	the second polygon

Returns

(Feature|null) - returns a feature representing the point(s) they share (in case of a Point or MultiPoint), the borders they share (in case of a LineString or a MultiLineString), the area they share (in case of Polygon or MultiPolygon). If they do not share any point, returns null.

Example

```
var poly1 = turf.polygon([[
  [-122.801742, 45.48565],
  [-122.801742, 45.60491],
  [-122.584762, 45.60491],
  [-122.584762, 45.48565],
  [-122.801742, 45.48565]
]]);

var poly2 = turf.polygon([[
  [-122.520217, 45.535693],
  [-122.64038, 45.553967],
  [-122.720031, 45.526554],
  [-122.669906, 45.507309],
  [-122.723464, 45.446643],
  [-122.532577, 45.408574],
  [-122.487258, 45.477466],
  [-122.520217, 45.535693]
]]);

var intersection = turf.intersect(poly1, poly2);
```

lineOffset

npm install @turf/line-offset

Takes a line and returns a line at offset by the specified distance.

Arguments

Argument	Type	Description
geojson	(Geometry|Feature <(LineString|MultiLineString)>)	input GeoJSON
distance	number	distance to offset the line (can be of negative value)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, kilometers, inches, yards, meters

Returns

Feature <(LineString|MultiLineString)> - Line offset from the input line

Example

```
var line = turf.lineString([[-83, 30], [-84, 36], [-78, 41]]

var offsetLine = turf.lineOffset(line, 2, {units: 'miles'});
```

simplify

npm install @turf/simplify

Takes a GeoJSON object and returns a simplified version. Internally uses simplify-js to perform

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

simplification using the Ramer-Douglas-Peucker algorithm.

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be simplified
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
tolerance	number	1	simplification tolerance
highQuality	boolean	false	whether or not to spend more time to create a higher-quality simplification with a different algorithm
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - a simplified GeoJSON

Example

```
var geojson = turf.polygon([[
  [-70.603637, -33.399918],
  [-70.614624, -33.395332],
  [-70.639343, -33.392466],
  [-70.659942, -33.394759],
  [-70.683975, -33.404504],
  [-70.697021, -33.419406],
  [-70.701141, -33.434306],
  [-70.700454, -33.446339],
  [-70.694274, -33.458369],
```

```
  [-70.682601, -33.465816],
  [-70.668869, -33.472117],
  [-70.646209, -33.473835],
  [-70.624923, -33.472117],
  [-70.609817, -33.468107],
  [-70.595397, -33.458369],
  [-70.587158, -33.442901],
  [-70.587158, -33.426283],
  [-70.590591, -33.414248],
  [-70.594711, -33.406224],
  [-70.603637, -33.399918]
]]);
var options = {tolerance: 0.01, highQuality: false};
var simplified = turf.simplify(geojson, options);
```

tesselate

npm install @turf/tesselate

Tesselates a Feature into a FeatureCollection of triangles using earcut.

Arguments

Argument	Type	Description
poly	Feature <Polygon>	the polygon to tesselate

Returns

FeatureCollection <Polygon> - a geometrycollection feature

Example

```
var poly = turf.polygon([[[11, 0], [22, 4], [31, 0], [31, 11]
var triangles = turf.tesselate(poly);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

transformRotate

npm install @turf/transform-rotate

Rotates any geojson Feature or Geometry of a specified angle, around its centroid or a given pivot point; all rotations follow the right-hand rule: https://en.wikipedia.org/wiki/Right-hand_rule

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be rotated
angle	number	of rotation (along the vertical axis), from North in decimal degrees, negative clockwise
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
pivot	Coord	'centroid'	point around which the rotation will be performed
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - the rotated GeoJSON feature

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var options = {pivot: [0, 25]};
var rotatedPoly = turf.transformRotate(poly, 10, options);
```

transformTransla

npm install @turf/transform-translate

Moves any geojson Feature or Geometry of a specified distance along a Rhumb Line on the provided direction angle.

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be translated
distance	number	length of the motion; negative values determine motion in opposite direction
direction	number	of the motion; angle from North in decimal degrees, positive clockwise
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	in which
zTranslation	number	0	length of the vertical motion, same unit of distance
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - the translated GeoJSON object

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var translatedPoly = turf.transformTranslate(poly, 100, 35);
```

transformScale

npm install @turf/transform-scale

Scale a GeoJSON from a given point by a factor of scaling (ex: factor=2 would make the GeoJSON 200% larger). If a FeatureCollection is provided, the origin point will be calculated based on each individual Feature.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be scaled
factor	number	of scaling, positive or negative values greater than 0
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
origin	(string|Coord)	'centroid'	Point from which the scaling will occur (string options: sw/se/nw/ne/center/centroid)
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - scaled GeoJSON

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var scaledPoly = turf.transformScale(poly, 3);
```

union

npm install @turf/union

Takes two or more polygons and returns a combined polygon. If the input polygons are not contiguous, this function returns a MultiPolygon feature.

Arguments

Argument	Type	Description
A	...Feature <Polygon>	polygon to combine

Returns

Feature <(Polygon|MultiPolygon)> - a combined Polygon or MultiPolygon feature

Example

```
var poly1 = turf.polygon([[
    [-82.574787, 35.594087],
    [-82.574787, 35.615581],
    [-82.545261, 35.615581],
    [-82.545261, 35.594087],
    [-82.574787, 35.594087]
]], {"fill": "#0f0"});
var poly2 = turf.polygon([[
    [-82.560024, 35.585153],
    [-82.560024, 35.602602],
    [-82.52964, 35.602602],
    [-82.52964, 35.585153],
    [-82.560024, 35.585153]
]], {"fill": "#00f"});

var union = turf.union(poly1, poly2);
```

voronoi

npm install @turf/voronoi

Takes a FeatureCollection of points, and a bounding box, and returns a FeatureCollection of Voronoi polygons.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to find the Voronoi polygons around.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
bbox	Array	[-180,-85,180,-85]	clipping rectangle, in

Returns

FeatureCollection <Polygon> - a set of polygons, one per input point.

Example

```
var options = {
  bbox: [-70, 40, -60, 60]
};
var points = turf.randomPoint(100, options);
var voronoiPolygons = turf.voronoi(points, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

combine

npm install @turf/combine

Combines a FeatureCollection of Point , LineString , or Polygon features into MultiPoint , MultiLineString , or MultiPolygon features.

Arguments

Argument	Type	Description
fc	FeatureCollection <(Point|LineString|Polygon)>	a FeatureCollection of any type

Returns

FeatureCollection <(MultiPoint|MultiLineString|MultiPolygon)> - a FeatureCollection of corresponding type to input

Example

```
var fc = turf.featureCollection([
  turf.point([19.026432, 47.49134]),
  turf.point([19.074497, 47.509548])
]);

var combined = turf.combine(fc);
```

explode

npm install @turf/explode

Takes a feature or set of features and returns all positions as points.

Arguments

Argument	Type	Description
geojson	GeoJSON	input features

Returns

FeatureCollection <point> - points representing the exploded input features

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Throws

Error - if it encounters an unknown geometry type

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31]

var explode = turf.explode(polygon);
```

flatten

npm install @turf/flatten

Flattens any GeoJSON to a FeatureCollection inspired by geojson-flatten.

Arguments

Argument	Type	Description
geojson	GeoJSON	any valid GeoJSON Object

Returns

FeatureCollection - all Multi-Geometries are flattened into single Features

Example

```
var multiGeometry = turf.multiPolygon([
  [[[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0],
  [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
  [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [
]);

var flatten = turf.flatten(multiGeometry);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lineToPolygon

npm install @turf/line-to-polygon

Converts (Multi)LineString(s) to Polygon(s).

Arguments

Argument	Type	Description
lines	(FeatureCollection|Feature <(LineString|MultiLineString)>)	Features to convert
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	translates GeoJSON properties to Feature
autoComplete	boolean	true	auto complete linestrings (matches first & last coordinates)
orderCoords	boolean	true	sorts linestrings to place outer ring at the first position of the coordinates

Returns

Feature <(Polygon|MultiPolygon)> - converted to Polygons

Example

```
var line = turf.lineString([[125, -30], [145, -30], [145, -2(

var polygon = turf.lineToPolygon(line);
```

polygonize

npm install @turf/polygonize

Polygonizes (Multi)LineString(s) into Polygons.

Arguments

Argument	Type	Description
geoJson	(FeatureCollection|Geometry|Feature <(LineString|MultiLineString)>)	Lines in order to polygonize

Returns

FeatureCollection <Polygon> - Polygons created

Throws

Error - if geoJson is invalid.

polygonToLine

npm install @turf/polygon-to-line

Converts a Polygon to (Multi)LineString or MultiPolygon to a FeatureCollection of (Multi)LineString.

Arguments

Argument	Type	Description
polygon	Feature <(Polygon|MultiPolygon)>	Feature to convert
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
properties	Object	{}	translates GeoJSON properties to Feature

Returns

(FeatureCollection|Feature <(LineString|MultiLinestring)>) - converted (Multi)Polygon to (Multi)LineString

Example

```
var poly = turf.polygon([[[125, -30], [145, -30], [145, -20]

var line = turf.polygonToLine(poly);
```

kinks

npm install @turf/kinks

Takes a linestring , multi-linestring , multi-polygon , or polygon and returns points at all self-intersections.

Arguments

Argument	Type	Description
featureIn	Feature <(LineString|MultiLineString|MultiPolygon|Polygon)>	input feature

Returns

FeatureCollection <Point> - self-intersections

Example

```
var poly = turf.polygon([[
  [-12.034835, 8.901183],
  [-12.060413, 8.899826],
  [-12.03638, 8.873199],
  [-12.059383, 8.871418],
```

```
  [-12.034835, 8.901183]
]]);

var kinks = turf.kinks(poly);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lineArc

npm install @turf/line-arc

Creates a circular arc, of a circle of the given radius and center point, between bearing1 and bearing2; 0 bearing is North of center point, positive clockwise.

Arguments

Argument	Type	Description
center	Coord	center point
radius	number	radius of the circle
bearing1	number	angle, in decimal degrees, of the first radius of the arc
bearing2	number	angle, in decimal degrees, of the second radius of the arc
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
steps	number	64	number of steps
units	string	'kilometers'	miles, kilometers, degrees, or radians

Returns

Feature <LineString> - line arc

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
var center = turf.point([-75, 40]);
var radius = 5;
var bearing1 = 25;
var bearing2 = 47;

var arc = turf.lineArc(center, radius, bearing1, bearing2);
```

lineChunk

npm install @turf/line-chunk

Divides a LineString into chunks of a specified length. If the line is shorter than the segment length then the original line is returned.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Geometry|Feature<(LineString|MultiLineString)>)	the lines to split
segmentLength	number	how long to make each segment
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	units can be degrees, radians, miles, or kilometers

Prop	Type	Default	Description
reverse	boolean	false	reverses coordinates to start the first chunked segment at the end

Returns

FeatureCollection <LineString> - collection of line segments

Example

```
var line = turf.lineString([[-95, 40], [-93, 45], [-85, 50]]]

var chunk = turf.lineChunk(line, 15, {units: 'miles'});
```

lineIntersect

npm install @turf/line-intersect

Takes any LineString or Polygon GeoJSON and returns the intersecting point(s).

Arguments

Argument	Type	Description
line1	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
line2	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon

Returns

FeatureCollection <Point> - point(s) that intersect both

Example

```
var line1 = turf.lineString([[126, -11], [129, -21]]);
var line2 = turf.lineString([[123, -18], [131, -14]]);
var intersects = turf.lineIntersect(line1, line2);
```

lineOverlap

npm install @turf/line-overlap

Takes any LineString or Polygon and returns the overlapping lines between both features.

Arguments

Argument	Type	Description
line1	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
line2	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
tolerance	number	0	Tolerance distance to match overlapping line segments (in kilometers)

Returns

FeatureCollection <LineString> - lines(s) that are overlapping between both features

Example

```
var line1 = turf.lineString([[115, -35], [125, -30], [135, -
var line2 = turf.lineString([[115, -25], [125, -30], [135, -

var overlapping = turf.lineOverlap(line1, line2);
```

lineSegment

npm install @turf/line-segment

Creates a FeatureCollection of 2-vertex LineString segments from a (Multi)LineString or (Multi)Polygon.

Arguments

Argument	Type	Description
geojson	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|MultiPolygon|Polygon)>)	GeoJSON Polygon or LineString

Returns

FeatureCollection <LineString> - 2-vertex line segments

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10
var segments = turf.lineSegment(polygon);
```

lineSlice

npm install @turf/line-slice

Takes a line , a start Point , and a stop point and returns a subsection of the line in-between those

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

points. The start & stop points don't need to fall exactly on the line.

Arguments

Argument	Type	Description
startPt	Coord	starting point
stopPt	Coord	stopping point
line	(Feature <LineString>|LineString)	line to slice

Returns

Feature <LineString> - sliced line

Example

```
var line = turf.lineString([
    [-77.031669, 38.878605],
    [-77.029609, 38.881946],
    [-77.020339, 38.884084],
    [-77.025661, 38.885821],
    [-77.021884, 38.889563],
    [-77.019824, 38.892368]
]);
var start = turf.point([-77.029609, 38.881946]);
var stop = turf.point([-77.021884, 38.889563]);

var sliced = turf.lineSlice(start, stop, line);
```

lineSliceAlong

npm install @turf/line-slice-along

Takes a line , a specified distance along the line to a start Point , and a specified distance along the line to a stop point and returns a subsection of the line in-between those points.

Arguments

Argument	Type	Description

Argument	Type	Description
line	(Feature <LineString>|LineString)	input line
startDist	number	distance along the line to starting point
stopDist	number	distance along the line to ending point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

Feature <LineString> - sliced line

Example

```
var line = turf.lineString([[7, 45], [9, 45], [14, 40], [14,
var start = 12.5;
var stop = 25;
var sliced = turf.lineSliceAlong(line, start, stop, {units:
```

lineSplit

npm install @turf/line-split

Split a LineString by another GeoJSON Feature.

Arguments

Argument	Type	Description
line	Feature <LineString>	LineString Feature to split
splitter	Feature	Feature used to split line

Returns

FeatureCollection <LineString> - Split LineStrings

Example

```
var line = turf.lineString([[120, -25], [145, -25]]);
var splitter = turf.lineString([[130, -15], [130, -35]]);

var split = turf.lineSplit(line, splitter);
```

mask

npm install @turf/mask

Takes any type of polygon and an optional mask and returns a polygon exterior ring with holes.

Arguments

Argument	Type	Description
polygon	(FeatureCollection|Feature <(Polygon|MultiPolygon)>)	GeoJSON Polygon used as interior rings or holes.
mask	(Feature <Polygon>)	GeoJSON Polygon used as the exterior ring (if undefined, the world extent is used)

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Returns

Feature <Polygon> - Masked Polygon (exterior ring with holes).

Example

```
var polygon = turf.polygon([[[112, -21], [116, -36], [146, -
var mask = turf.polygon([[[90, -55], [170, -55], [170, 10],

var masked = turf.mask(polygon, mask);
```

nearestPointOnL

npm install @turf/nearest-point-on-line

Takes a Point and a LineString and calculates the closest Point on the (Multi)LineString.

Arguments

Argument	Type	Description
lines	(Geometry|Feature <(LineString|MultiLineString)>)	lines to snap to
pt	(Geometry|Feature <Point>|Array)	point to snap from
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

Feature <Point> - closest point on the line to point. The properties object will contain three

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

values: index : closest point was found on nth line part, dist : distance between pt and the closest point, location : distance along the line between start and the closest point.

Example

```
var line = turf.lineString([
    [-77.031669, 38.878605],
    [-77.029609, 38.881946],
    [-77.020339, 38.884084],
    [-77.025661, 38.885821],
    [-77.021884, 38.889563],
    [-77.019824, 38.892368]
]);
var pt = turf.point([-77.037076, 38.884017]);

var snapped = turf.nearestPointOnLine(line, pt, {units: 'mil
```

sector

npm install @turf/sector

Creates a circular sector of a circle of given radius and center Point , between (clockwise) bearing1 and bearing2; 0 bearing is North of center point, positive clockwise.

Arguments

Argument	Type	Description
center	Coord	center point
radius	number	radius of the circle
bearing1	number	angle, in decimal degrees, of the first radius of the sector
bearing2	number	angle, in decimal degrees, of the second radius of the sector
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	miles, kilometers, degrees, or radians
steps	number	64	number of steps
properties	Properties	{}	Translate properties to Feature Polygon

Returns

Feature <Polygon> - sector polygon

Example

```
var center = turf.point([-75, 40]);
var radius = 5;
var bearing1 = 25;
var bearing2 = 45;

var sector = turf.sector(center, radius, bearing1, bearing2);
```

shortestPath

npm install @turf/shortest-path

Returns the shortest path from start to end without colliding with any Feature in obstacles

Arguments

Argument	Type	Description
start	Coord	point
end	Coord	point
options	Object	optional parameters

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Options

Prop	Type	Default	Description
obstacles	((Geometry|Feature|FeatureCollection <Polygon>))		areas which path cannot travel
minDistance	(number)		minimum distance between shortest path and obstacles
units	string	'kilometers'	unit in which resolution & minimum distance will be expressed in; it can be degrees, radians, miles, kilometers, ...
resolution	number	100	distance between matrix points on which the path will be calculated

Returns

Feature <LineString> - shortest path between start and end

Example

```
var start = [-5, -6];
var end = [9, -6];
var options = {
  obstacles: turf.polygon([[[0, -7], [5, -7], [5, -3], [0, -
};

var path = turf.shortestPath(start, end, options);
```

unkinkPolygon

npm install @turf/unkink-polygon

Takes a kinked polygon and returns a feature collection of polygons that have no kinks. Uses simplepolygon internally.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature<(Polygon|MultiPolygon)>)	GeoJSON Polygon or MultiPolygon

Returns

FeatureCollection <Polygon> - Unkinked polygons

Example

```
var poly = turf.polygon([[[0, 0], [2, 0], [0, 2], [2, 2], [0,

var result = turf.unkinkPolygon(poly);
```

featureCollectior

npm install @turf/helpers

Takes one or more Features and creates a FeatureCollection.

Note: featureCollection is part of the @turf/helpers module.

To use it as a stand-alone

Arguments

Argument	Type	Description
features	Array<Feature>	input features

Argument	Type	Description
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

FeatureCollection - FeatureCollection of Features

Example

```
var locationA = turf.point([-75.343, 39.984], {name: 'Locati
var locationB = turf.point([-75.833, 39.284], {name: 'Locati
var locationC = turf.point([-75.534, 39.123], {name: 'Locati

var collection = turf.featureCollection([
  locationA,
  locationB,
  locationC
]);

//=collection
```

module will need to import @turf/helpers and call the featureCollection method.

feature

npm install @turf/helpers

Wraps a GeoJSON Geometry in a GeoJSON Feature.

Arguments

Argument	Type	Description
geometry	Geometry	input geometry

Note: feature is part of the @turf/helpers module.

To use it as a stand-alone

Argument	Type	Description
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

module will need to import @turf/helpers and call the feature method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature - a GeoJSON Feature

Example

```
var geometry = {
  "type": "Point",
  "coordinates": [110, 50]
};

var feature = turf.feature(geometry);

//=feature
```

geometryCollect

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
geometries	Array <Geometry>	an array of GeoJSON Geometries

Note: geometryCollection is part of the @turf/helpers module.

To use it as a stand-alone

Argument	Type	Description
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

module will need to import @turf/helpers and call the geometryCollection method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <GeometryCollection> - a GeoJSON GeometryCollection Feature

Example

```
var pt = {
    "type": "Point",
      "coordinates": [100, 0]
    };
var line = {
    "type": "LineString",
    "coordinates": [ [101, 0], [102, 1] ]
  };
var collection = turf.geometryCollection([pt, line]);

//=collection
```

lineString

npm install @turf/helpers

Creates a LineString Feature from an Array of Positions.

Arguments

Note: lineString is part of the @turf/helpers module.

Argument	Type	Description
coordinates	Array >	an array of Positions
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

To use it as a stand-alone module will need to import @turf/helpers and call the lineString method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <LineString> - LineString Feature

Example

```
var linestring1 = turf.lineString([[-24, 63], [-23, 60], [-25
var linestring2 = turf.lineString([[-14, 43], [-13, 40], [-15

//=linestring1
//=linestring2
```

multiLineString

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >>	an array of LineStrings
properties	Object	an Object of key-value pairs to add as properties

Note: multiLineString is part of the @turf/helpers module.

To use it as a stand-alone module will need

Argument	Type	Description
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiLineString> - a MultiLineString feature

Throws

Error - if no coordinates are passed

Example

```
var multiLine = turf.multiLineString([[[0,0],[10,10]]]);

//=multiLine
```

to import @turf/helpers and call the multiLineString method.

multiPoint

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >	an array of Positions
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Note: multiPoint is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the multiPoint method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiPoint> - a MultiPoint feature

Throws

Error - if no coordinates are passed

Example

```
var multiPt = turf.multiPoint([[0,0],[10,10]]);

//=multiPt
```

multiPolygon

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >>>	an array of Polygons
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description

Note: multiPolygon is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the multiPolygon method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiPolygon> - a multipolygon feature

Throws

Error - if no coordinates are passed

Example

```
var multiPoly = turf.multiPolygon([[[[0,0],[0,10],[10,10],[10

//=multiPoly
```

point

npm install @turf/helpers

Creates a Point Feature from a Position.

Note: point is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the point method.

Arguments

Argument	Type	Description
coordinates	Array	longitude, latitude position (each in decimal degrees)
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <Point> - a Point feature

Example

```
var point = turf.point([-75.343, 39.984]);

//=point
```

polygon

npm install @turf/helpers

Creates a Polygon Feature from an Array of LinearRings.

Note: polygon is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the polygon method.

Arguments

Argument	Type	Description
coordinates	Array >>	an array of LinearRings
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <Polygon> - Polygon Feature

Example

```
var polygon = turf.polygon([[[-5, 52], [-4, 56], [-2, 51], [

//=polygon
```

randomPosition

npm install @turf/random

Returns a random position within a box.

Arguments

Argument	Type	Description
bbox	Array	a bounding box inside of which positions are placed.

Returns

Array - Position longitude, latitude

Note: randomPosition is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPosition method.

Example

```
var position = turf.randomPosition([-180, -90, 180, 90])
//=position
```

randomPoint

npm install @turf/random

Returns a random point.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
bbox	Array	[-180,-90,180,90]	a bounding box inside of which geometries are placed.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

```
var points = turf.randomPoint(25, {bbox: [-180, -90, 180, 90]
//=points
```

Note: randomPoint is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPoint method.

randomLineStrir

npm install @turf/random

Returns a random linestring.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Note: randomLineString is part of the @turf/random module.

To use it as a stand-alone module will need to import

Options

Prop	Type	Default	Description
bbox	Array	[-180,-90,180,90]	a bounding box inside of which geometries are placed.
num_vertices	number	10	is how many coordinates each LineString will contain.
max_length	number	0.0001	is the maximum number of decimal degrees that a vertex can be from its predecessor
max_rotation	number	Math.PI/8	is the maximum number of radians that a line segment can turn from the previous segment.

@turf/random and call the randomLineString method.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

```
var lineStrings = turf.randomLineString(25, {bbox: [-180, -9
//=lineStrings
```

randomPolygon

npm install @turf/random

Returns a random polygon.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Descript
bbox	Array	[-180,-90,180,90]	a bound box insi of which geometries are placed.
num_vertices	number	10	is how many coordinates each LineString will contain.
max_radial_length	number	10	is the maximum number of decimal degrees latitude or longitude that a vertex can reach out of the center of the Polygon.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

Note: randomPolygon is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPolygon method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var polygons = turf.randomPolygon(25, {bbox: [-180, -90, 180
//=polygons
```

sample

npm install @turf/sample

Takes a FeatureCollection and returns a FeatureCollection with given number of features at random.

Arguments

Argument	Type	Description
featurecollection	FeatureCollection	set of input features
num	number	number of features to select

Returns

FeatureCollection - a FeatureCollection with n features

Example

```
var points = turf.randomPoint(100, {bbox: [-80, 30, -60, 60]}

var sample = turf.sample(points, 5);
```

interpolate

npm install @turf/interpolate

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Takes a set of points and estimates their 'property' values on a grid using the Inverse Distance Weighting (IDW) method.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	with known value
cellSize	number	the distance across each grid point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
gridType	string	'square'	defines the output format based on a Grid Type (options: 'square' | 'point' | 'hex' | 'triangle')
property	string	'elevation'	the property name in
units	string	'kilometers'	used in calculating cellSize, can be degrees, radians, miles, or kilometers
weight	number	1	exponent regulating the distance-decay weighting

Returns

FeatureCollection <(Point|Polygon)> - grid of points or polygons with interpolated 'property'

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
var points = turf.randomPoint(30, {bbox: [50, 30, 70, 50]});

// add a random property to each point
turf.featureEach(points, function(point) {
    point.properties.solRad = Math.random() * 50;
});
var options = {gridType: 'points', property: 'solRad', units
var grid = turf.interpolate(points, 100, options);
```

isobands

npm install @turf/isobands

Takes a grid FeatureCollection of Point features with z-values and an array of value breaks and generates filled contour isobands.

Arguments

Argument	Type	Description
pointGrid	FeatureCollection <Point>	input points
breaks	Array	where to draw contours
options	Object	options on output

Options

Prop	Type	Default	Description
zProperty	string	'elevation'	the property name in
commonProperties	Object	{}	GeoJSON properties passed to A isobands

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
breaksProperties	Array	[]	GeoJSON properties passed, in order, to the corresponde isoband (orc defined by breaks)

Returns

FeatureCollection <MultiPolygon> - a FeatureCollection of MultiPolygon features representing isobands

isolines

npm install @turf/isolines

Takes a grid FeatureCollection of Point features with z-values and an array of value breaks and generates isolines.

Arguments

Argument	Type	Description
pointGrid	FeatureCollection <Point>	input points
breaks	Array	values of zProperty where to draw isolines
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
zProperty	string	'elevation'	the property name in

Prop	Type	Default	Description
commonProperties	Object	{}	GeoJSON properties passed to A isolines
breaksProperties	Array	[]	GeoJSON properties passed, in order, to the corresponde isoline; the breaks array will define tl order in whi the isolines are created

Returns

FeatureCollection <MultiLineString> - a FeatureCollection of MultiLineString features representing isolines

Example

```
// create a grid of points with random z-values in their prop
var extent = [0, 30, 20, 50];
var cellWidth = 100;
var pointGrid = turf.pointGrid(extent, cellWidth, {units: 'm

for (var i = 0; i < pointGrid.features.length; i++) {
    pointGrid.features[i].properties.temperature = Math.rando
}
var breaks = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var lines = turf.isolines(pointGrid, breaks, {zProperty: 'ter
```

planepoint

npm install @turf/planepoint

Takes a triangular plane as a Polygon and a Point within that triangle and returns the z-value at that point. The Polygon should have properties a , b ,

and c that define the values at its three corners. Alternatively, the z-values of each triangle point can be provided by their respective 3rd coordinate if their values are not provided as properties.

Arguments

Argument	Type	Description
point	Coord	the Point for which a z-value will be calculated
triangle	Feature <Polygon>	a Polygon feature with three vertices

Returns

number - the z-value for interpolatedPoint

Example

```
var point = turf.point([-75.3221, 39.529]);
// "a", "b", and "c" values represent the values of the coord
var triangle = turf.polygon([[
  [-75.1221, 39.57],
  [-75.58, 39.18],
  [-75.97, 39.86],
  [-75.1221, 39.57]
]], {
  "a": 11,
  "b": 122,
  "c": 44
});

var zValue = turf.planepoint(point, triangle);
point.properties.zValue = zValue;
```

tin

npm install @turf/tin

Takes a set of points and creates a Triangulated Irregular Network , or a TIN for short, returned as a collection of Polygons. These are often used for

developing elevation contour maps or stepped heat visualizations.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	input points
z	(String)	name of the property from which to pull z values This is optional: if not given, then there will be no extra data added to the derived triangles.

Returns

FeatureCollection <Polygon> - TIN output

Example

```
// generate some random point data
var points = turf.randomPoint(30, {bbox: [50, 30, 70, 50]});

// add a random property to each point between 0 and 9
for (var i = 0; i < points.features.length; i++) {
  points.features[i].properties.z = ~~(Math.random() * 9);
}
var tin = turf.tin(points, 'z');
```

pointsWithinPoly

npm install @turf/points-within-polygon

Finds Points that fall within (Multi)Polygon(s).

Arguments

Argument	Type	Description
points	(Feauture|FeatureCollection <Point>)	Points as input search

Argument	Type	Description
polygons	(FeatureCollection|Geometry|Feature <(Polygon|MultiPolygon)>)	Points must be within these (Multi)Polygon(s)

Returns

FeatureCollection <Point> - points that land within at least one polygon

Example

```
var points = turf.points([
    [-46.6318, -23.5523],
    [-46.6246, -23.5325],
    [-46.6062, -23.5513],
    [-46.663, -23.554],
    [-46.643, -23.557]
]);

var searchWithin = turf.polygon([[
    [-46.653,-23.543],
    [-46.634,-23.5346],
    [-46.613,-23.543],
    [-46.614,-23.559],
    [-46.631,-23.567],
    [-46.653,-23.560],
    [-46.653,-23.543]
]]);

var ptsWithin = turf.pointsWithinPolygon(points, searchWithir
```

tag

npm install @turf/tag

Takes a set of points and a set of polygons and performs a spatial join.

Arguments

Argument	Type	Description

Argument	Type	Description
points	FeatureCollection <Point>	input points
polygons	FeatureCollection <Polygon>	input polygons
field	string	property in polygons to add to joined { } features
outField	string	property in points in which to store joined property from polygons

Returns

FeatureCollection <Point> - points with containingPolyId property containing values from polyId

Example

```
var pt1 = turf.point([-77, 44]);
var pt2 = turf.point([-77, 38]);
var poly1 = turf.polygon([[
  [-81, 41],
  [-81, 47],
  [-72, 47],
  [-72, 41],
  [-81, 41]
]], {pop: 3000});
var poly2 = turf.polygon([[
  [-81, 35],
  [-81, 41],
  [-72, 41],
  [-72, 35],
  [-81, 35]
]], {pop: 1000});

var points = turf.featureCollection([pt1, pt2]);
var polygons = turf.featureCollection([poly1, poly2]);

var tagged = turf.tag(points, polygons, 'pop', 'population')
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

hexGrid

npm install @turf/hex-grid

Takes a bounding box and the diameter of the cell and returns a FeatureCollection of flat-topped hexagons or triangles (Polygon features) aligned in an "odd-q" vertical grid as described in Hexagonal Grids.

Arguments

Argument	Type	Description
bbox	BBox	extent in minX, minY, maxX, maxY order
cellSide	number	length of the side of the the hexagons or triangles, in units. It will also coincide with the radius of the circumcircle of the hexagons.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cell size, can be degrees, radians, miles, or kilometers
properties	Object	{}	passed to each hexagon or triangle of the grid
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it

Prop	Type	Default	Description
triangles	boolean	false	whether to return as triangles instead of hexagons

Returns

FeatureCollection <Polygon> - a hexagonal grid

Example

```
var bbox = [-96,31,-84,40];
var cellSide = 50;
var options = {units: 'miles'};

var hexgrid = turf.hexGrid(bbox, cellSide, options);
```

pointGrid

npm install @turf/point-grid

Creates a Point grid from a bounding box, FeatureCollection or Feature.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order
cellSide	number	the distance between points, in units
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Point> - grid of points

Example

```
var extent = [-70.823364, -33.553984, -70.473175, -33.302986]
var cellSide = 3;
var options = {units: 'miles'};

var grid = turf.pointGrid(extent, cellSide, options);
```

squareGrid

npm install @turf/square-grid

Creates a square grid from a bounding box, Feature or FeatureCollection.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order

Argument	Type	Description
cellSide	number	of each cell, in units
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Polygon> - grid a grid of polygons

Example

```
var bbox = [-95, 30 ,-85, 40];
var cellSide = 50;
var options = {units: 'miles'};

var squareGrid = turf.squareGrid(bbox, cellSide, options);
```

triangleGrid

npm install @turf/triangle-grid

Takes a bounding box and a cell depth and returns a set of triangular polygons in a grid.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order
cellSide	number	dimension of each cell
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Polygon> - grid of polygons

Example

```
var bbox = [-95, 30 ,-85, 40];
var cellSide = 50;
var options = {units: 'miles'};

var triangleGrid = turf.triangleGrid(bbox, cellSide, options)
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

nearestPoint

npm install @turf/nearest-point

Takes a reference point and a FeatureCollection of Features with Point geometries and returns the point from the FeatureCollection closest to the reference. This calculation is geodesic.

Arguments

Argument	Type	Description
targetPoint	Coord	the reference point
points	FeatureCollection <Point>	against input point set

Returns

Feature <Point> - the closest point in the set to the reference point

Example

```
var targetPoint = turf.point([28.965797, 41.010086], {"marke
var points = turf.featureCollection([
    turf.point([28.973865, 41.011122]),
    turf.point([28.948459, 41.024204]),
    turf.point([28.938674, 41.013324])
]);

var nearest = turf.nearestPoint(targetPoint, points);
```

collect

npm install @turf/collect

Merges a specified property from a FeatureCollection of points into a FeatureCollection of polygons. Given an inProperty on points and an outProperty for polygons, this finds every point that lies within each polygon,

collects the inProperty values from those points, and adds them as an array to outProperty on the polygon.

Arguments

Argument	Type	Description
polygons	FeatureCollection <Polygon>	polygons with values on which to aggregate
points	FeatureCollection <Point>	points to be aggregated
inProperty	string	property to be nested from
outProperty	string	property to be nested into

Returns

FeatureCollection <Polygon> - polygons with properties listed based on outField

Example

```
var poly1 = turf.polygon([[[0,0],[10,0],[10,10],[0,10],[0,0]]
var poly2 = turf.polygon([[[10,0],[20,10],[20,20],[20,0],[10,
var polyFC = turf.featureCollection([poly1, poly2]);
var pt1 = turf.point([5,5], {population: 200});
var pt2 = turf.point([1,3], {population: 600});
var pt3 = turf.point([14,2], {population: 100});
var pt4 = turf.point([13,1], {population: 200});
var pt5 = turf.point([19,7], {population: 300});
var pointFC = turf.featureCollection([pt1, pt2, pt3, pt4, pt5
var collected = turf.collect(polyFC, pointFC, 'population',
var values = collected.features[0].properties.values
//=values => [200, 600]
```

clustersDbscan

npm install @turf/clusters-dbscan

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Takes a set of points and partition them into clusters according to https://en.wikipedia.org/wiki/DBSCAN data clustering algorithm.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to be clustered
maxDistance	number	Maximum Distance between any point of the cluster to generate the clusters (kilometers only)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	in which
mutate	boolean	false	Allows GeoJSON input to be mutated
minPoints	number	3	Minimum number of points to generate a single cluster, points which do not meet this requirement will be classified as an 'edge' or 'noise'.

Returns

FeatureCollection <Point> - Clustered Points with an additional two properties associated to each Feature:

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
// create random points with random z-values in their properi
var points = turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
var maxDistance = 100;
var clustered = turf.clustersDbscan(points, maxDistance);
```

clustersKmeans

npm install @turf/clusters-kmeans

Takes a set of points and partition them into clusters using the k-mean. It uses the k-means algorithm

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to be clustered
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
numberOfClusters	number	Math.sqrt(numberOfPoints/2)	numberOfClusters that will be generated
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

FeatureCollection <Point> - Clustered Points with an additional two properties associated to each Feature:

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
// create random points with random z-values in their properi
var points = turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
var options = {numberOfClusters: 7};
var clustered = turf.clustersKmeans(points, options);
```

coordAll

npm install @turf/meta

Get all coordinates from any GeoJSON object.

Arguments

Argument	Type	Descr
geojson	(FeatureCollection|Feature|Geometry)	any GeoJ objec

Returns

Array > - coordinate position array

Note: coordAll is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordAll method.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {foo: 'bar'}),
  turf.point([36, 53], {hello: 'world'})
]);

var coords = turf.coordAll(features);
//= [[26, 37], [36, 53]]
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

coordEach

npm install @turf/meta

Iterate over coordinates in any GeoJSON object, similar to Array.forEach()

Note: coordEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordEach method.

Arguments

Argument	Type	
geojson	(FeatureCollection|Feature|Geometry)	
callback	Function	
excludeWrapCoord	boolean	whether or not to include the final coordinate of LinearRings that wraps the ring in its iteration.

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.coordEach(features, function (currentCoord, coordIndex,
  //=currentCoord
  //=coordIndex
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

coordReduce

npm install @turf/meta

Reduce coordinates in any GeoJSON object, similar to Array.reduce()

Note: coordReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordReduce method.

Arguments

Argument	Type	
geojson	(FeatureCollection|Geometry|Feature)	
callback	Function	
initialValue	(*)	argument to the first call of the callback.
excludeWrapCoord	boolean	whether or not to include the final coordinate of LinearRings that wraps the ring in its iteration.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.coordReduce(features, function (previousValue, currentC
  //=previousValue
  //=currentCoord
  //=coordIndex
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
  return currentCoord;
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

featureEach

npm install @turf/meta

Iterate over features in any GeoJSON object, similar to Array.forEach.

Note: featureEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the featureEach method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {foo: 'bar'}),
  turf.point([36, 53], {hello: 'world'})
]);

turf.featureEach(features, function (currentFeature, featureI
  //=currentFeature
  //=featureIndex
});
```

featureReduce

npm install @turf/meta

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Reduce features in any GeoJSON object, similar to Array.reduce().

Note: featureReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the featureReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curre featu
initialValue	(*)	Valu the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.featureReduce(features, function (previousValue, current
  //=previousValue
  //=currentFeature
  //=featureIndex
  return currentFeature
});
```

flattenEach

npm install @turf/meta

Iterate over flattened features in any GeoJSON object, similar to Array.forEach.

Note: flattenEach is part of the

Arguments

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu multil

@turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the flattenEach method.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.multiPoint([[40, 30], [36, 53]], {hello: 'world'})
]);

turf.flattenEach(features, function (currentFeature, featureI
  //=currentFeature
  //=featureIndex
  //=multiFeatureIndex
});
```

flattenReduce

npm install @turf/meta

Reduce flattened features in any GeoJSON object, similar to Array.reduce().

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curre featu mult

Note: flattenReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the flattenReduce method.

Argument	Type	Description
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.multiPoint([[40, 30], [36, 53]], {hello: 'world'})
]);

turf.flattenReduce(features, function (previousValue, current
  //=previousValue
  //=currentFeature
  //=featureIndex
  //=multiFeatureIndex
  return currentFeature
});
```

getCoord

npm install @turf/invariant

Unwrap a coordinate from a Point Feature, Geometry or a single coordinate.

Arguments

Argument	Type	Description
coord	(Array |Geometry <Point>|Feature <Point>)	GeoJSON Point or an Array of numbers

Returns

Array - coordinates

Note: getCoord is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getCoord method.

Example

```
var pt = turf.point([10, 10]);

var coord = turf.getCoord(pt);
//= [10, 10]
```

getCoords

npm install @turf/invariant

Unwrap coordinates from a Feature, Geometry Object or an Array

Note: getCoords is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getCoords method.

Arguments

Argument	Type	Description
coords	(Array |Geometry|Feature)	Feature, Geometry Object or an Array

Returns

Array - coordinates

Example

```
var poly = turf.polygon([[[119.32, -8.7], [119.55, -8.69], [1

var coords = turf.getCoords(poly);
//= [[[119.32, -8.7], [119.55, -8.69], [119.51, -8.54], [119.
```

getGeom

npm install @turf/invariant

Get Geometry from Feature or Geometry Object

Note: getGeom is part of the

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
geojson	(Feature|Geometry)	GeoJSON Feature or Geometry Object

Returns

(Geometry|null) - GeoJSON Geometry Object

Throws

Error - if geojson is not a Feature or Geometry Object

Example

```
var point = {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [110, 40]
  }
}
var geom = turf.getGeom(point)
//={"type": "Point", "coordinates": [110, 40]}
```

@turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getGeom method.

getType

Get GeoJSON object's type, Geometry type is prioritize.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON object
name	string	name of the variable to display in error message

Returns

string - GeoJSON type

Example

npm install @turf/invariant

Note: getType is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getType method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var point = {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [110, 40]
  }
}
var geom = turf.getType(point)
//="Point"
```

geomEach

npm install @turf/meta

Iterate over each geometry in any GeoJSON object, similar to Array.forEach()

Note: geomEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the geomEach method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu featu featu featu

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.geomEach(features, function (currentGeometry, featureIn
  //=currentGeometry
  //=featureIndex
```

```
  //=featureProperties
  //=featureBBox
  //=featureId
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

geomReduce

npm install @turf/meta

Reduce geometry in any GeoJSON object, similar to Array.reduce().

Note: geomReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the geomReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curre featu featu featu featu
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.geomReduce(features, function (previousValue, currentGe
  //=previousValue
  //=currentGeometry
```

```
  //=featureIndex
  //=featureProperties
  //=featureBBox
  //=featureId
  return currentGeometry
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

propEach

npm install @turf/meta

Iterate over properties in any GeoJSON object, similar to Array.forEach()

Note: propEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the propEach method.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature)	any GeoJSON object
callback	Function	a method that takes (currentProperti featureIndex)

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.propEach(features, function (currentProperties, featureI
  //=currentProperties
  //=featureIndex
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

propReduce

npm install @turf/meta

Reduce properties in any GeoJSON object into a single value, similar to how Array.reduce works. However, in this case we lazily run the reduction, so an array of all properties is unnecessary.

Note: propReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the propReduce method.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature)	any GeoJSON object
callback	Function	a method that takes (previousValue currentPropert featureIndex)
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.propReduce(features, function (previousValue, currentPr
  //=previousValue
  //=currentProperties
  //=featureIndex
  return currentProperties
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

segmentEach

npm install @turf/meta

Iterate over 2-vertex line segment in any GeoJSON object, similar to Array.forEach() (Multi)Point geometries do not contain segments therefore they are ignored during this operation.

Note: segmentEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the segmentEach method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G
callback	Function	a met takes (curre featu multil geom segmentIndex)

Returns

undefined - undefined

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10]

// Iterate over GeoJSON by 2-vertex segments
turf.segmentEach(polygon, function (currentSegment, featureIn
  //=currentSegment
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
  //=segmentIndex
});

// Calculate the total number of segments
var total = 0;
turf.segmentEach(polygon, function () {
    total++;
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

segmentReduce

npm install @turf/meta

Reduce 2-vertex line segment in any GeoJSON object, similar to Array.reduce() (Multi)Point geometries do not contain segments therefore they are ignored during this operation.

Note: segmentReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the segmentReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any
callback	Function	a me take (prev curre curre
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

undefined - undefined

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10]

// Iterate over GeoJSON by 2-vertex segments
turf.segmentReduce(polygon, function (previousSegment, currer
  //= previousSegment
  //= currentSegment
  //= featureIndex
  //= multiFeatureIndex
  //= geometryIndex
  //= segmentInex
  return currentSegment
});

// Calculate the total number of segments
var initialValue = 0
var total = turf.segmentReduce(polygon, function (previousVal
    previousValue++;
    return previousValue;
}, initialValue);
```

getCluster

npm install @turf/clusters

Get Cluster

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
filter	*	Filter used on GeoJSON properties to get Cluster

Note: getCluster is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the getCluster method.

Returns

FeatureCollection - Single Cluster filtered by GeoJSON Properties

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0], {'marker-symbol': 'circle'}),
    turf.point([2, 4], {'marker-symbol': 'star'}),
    turf.point([3, 6], {'marker-symbol': 'star'}),
    turf.point([5, 1], {'marker-symbol': 'square'}),
    turf.point([4, 2], {'marker-symbol': 'circle'})
]);

// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Retrieve first cluster (0)
var cluster = turf.getCluster(clustered, {cluster: 0});
//= cluster

// Retrieve cluster based on custom properties
turf.getCluster(clustered, {'marker-symbol': 'circle'}).lengt
//= 2
turf.getCluster(clustered, {'marker-symbol': 'square'}).lengt
//= 1
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

clusterEach

npm install @turf/clusters

clusterEach

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
property	(string|number)	GeoJSON property key/value used to create clusters
callback	Function	a method that takes (cluster, clusterValue, currentIndex)

Note: clusterEach is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the clusterEach method.

Returns

undefined - undefined

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0]),
    turf.point([2, 4]),
    turf.point([3, 6]),
    turf.point([5, 1]),
    turf.point([4, 2])
]);

// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Iterate over each cluster
turf.clusterEach(clustered, 'cluster', function (cluster, clu
    //= cluster
    //= clusterValue
    //= currentIndex
})

// Calculate the total number of clusters
var total = 0
turf.clusterEach(clustered, 'cluster', function () {
    total++;
});

// Create an Array of all the values retrieved from the 'clus
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var values = []
turf.clusterEach(clustered, 'cluster', function (cluster, clu
    values.push(clusterValue);
});
```

clusterReduce

npm install @turf/clusters

Reduce clusters in GeoJSON Features, similar to Array.reduce()

Note: clusterReduce is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the clusterReduce method.

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
property	(string|number)	GeoJSON property key/value used to create clusters
callback	Function	a method that takes (previousValue, cluster, clusterValue, currentIndex)
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0]),
    turf.point([2, 4]),
    turf.point([3, 6]),
    turf.point([5, 1]),
    turf.point([4, 2])
]);
```

```
// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Iterate over each cluster and perform a calculation
var initialValue = 0
turf.clusterReduce(clustered, 'cluster', function (previousVa
    //=previousValue
    //=cluster
    //=clusterValue
    //=currentIndex
    return previousValue++;
}, initialValue);

// Calculate the total number of clusters
var total = turf.clusterReduce(clustered, 'cluster', functio
    return previousValue++;
}, 0);

// Create an Array of all the values retrieved from the 'clus
var values = turf.clusterReduce(clustered, 'cluster', functi
    return previousValue.concat(clusterValue);
}, []);
```

collectionOf

npm install @turf/invariant

Enforce expectations about types of FeatureCollection inputs for Turf. Internally this uses geojsonType to judge geometry types.

Note: collectionOf is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the collectionOf method.

Arguments

Argument	Type	Description
featureCollection	FeatureCollection	a FeatureCollection for which features will be judged
type	string	expected GeoJSON type
name	string	name of calling function

TURF

Throws

Error - if value is not the expected type.

containsNumber

npm install @turf/invariant

Checks if coordinates contains a number

Arguments

Argument	Type	Description
coordinates	Array	GeoJSON Coordinates

Returns

boolean - true if Array contains a number

Note: containsNumber is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the containsNumber method.

geojsonType

npm install @turf/invariant

Enforce expectations about types of GeoJSON objects for Turf.

Arguments

Argument	Type	Description
value	GeoJSON	any GeoJSON object
type	string	expected GeoJSON type
name	string	name of calling function

Throws

Error - if value is not the expected type.

Note: geojsonType is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

geojsonType method.

featureOf

npm install @turf/invariant

Enforce expectations about types of Feature inputs for Turf. Internally this uses geojsonType to judge geometry types.

Note: featureOf is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the featureOf method.

Arguments

Argument	Type	Description
feature	Feature	a feature with an expected geometry type
type	string	expected GeoJSON type
name	string	name of calling function

Throws

Error - error if value is not the expected type.

booleanClockwis

npm install @turf/boolean-clockwise

Takes a ring and return true or false whether or not the ring is clockwise or counter-clockwise.

Arguments

Argument	Type	Description
line	Feature <LineString>	to be evaluated

Returns

boolean - true/false

Example

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var clockwiseRing = turf.lineString([[0,0],[1,1],[1,0],[0,0]]
var counterClockwiseRing = turf.lineString([[0,0],[1,0],[1,1]

turf.booleanClockwise(clockwiseRing)
//=true
turf.booleanClockwise(counterClockwiseRing)
//=false
```

booleanContains

npm install @turf/boolean-contains

Boolean-contains returns True if the second geometry is completely contained by the first geometry. The interiors of both geometries must intersect and, the interior and boundary of the secondary (geometry b) must not intersect the exterior of the primary (geometry a). Boolean-contains returns the exact opposite result of the @turf/boolean-within.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);
var point = turf.point([1, 2]);

turf.booleanContains(line, point);
//=true
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

booleanCrosses

npm install @turf/boolean-crosses

Boolean-Crosses returns True if the intersection results in a geometry whose dimension is one less than the maximum dimension of the two source geometries and the intersection set is interior to both source geometries.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line1 = turf.lineString([[-2, 2], [4, 2]]);
var line2 = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]

var cross = turf.booleanCrosses(line1, line2);
//=true
```

booleanDisjoint

npm install @turf/boolean-disjoint

Boolean-disjoint returns (TRUE) if the intersection of the two geometries is an empty set.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry

Argument	Type	Description
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var point = turf.point([2, 2]);
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);

turf.booleanDisjoint(line, point);
//=true
```

booleanEqual

npm install @turf/boolean-equal

Determine whether two geometries of the same type have identical X,Y coordinate values. See http://edndoc.esri.com/arcsde/9.0/general_topics/understand_spatial_relations.htm

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON input
feature2	(Geometry|Feature)	GeoJSON input

Returns

boolean - true if the objects are equal, false otherwise

Example

```
var pt1 = turf.point([0, 0]);
var pt2 = turf.point([0, 0]);
var pt3 = turf.point([1, 1]);

turf.booleanEqual(pt1, pt2);
//= true
```

```
turf.booleanEqual(pt2, pt3);
//= false
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
- along
- area
- bbox
- bboxPolygon
- bearing
- center
- centerOfMass
- centroid
- destination
- distance
- envelope
- length
- midpoint
- pointOnFeature
- polygonTangents
- pointToLineDista
- rhumbBearing
- rhumbDestinatio
- rhumbDistance
- square
- greatCircle

COORDINATE MUTATION
- cleanCoords
- flip
- rewind
- round
- truncate

TRANSFORMATIO
- bboxClip
- bezierSpline
- buffer
- circle
- clone
- concave
- convex
- difference
- dissolve
- intersect
- lineOffset

booleanOverlap

npm install @turf/boolean-overlap

Compares two geometries of the same dimension and returns true if their intersection set results in a geometry different from both but of the same dimension. It applies to Polygon/Polygon, LineString/LineString, Multipoint/Multipoint, MultiLineString/MultiLineString and MultiPolygon/MultiPolygon.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	input
feature2	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	input

Returns

boolean - true/false

Example

```
var poly1 = turf.polygon([[[0,0],[0,5],[5,5],[5,0],[0,0]]]);
var poly2 = turf.polygon([[[1,1],[1,6],[6,6],[6,1],[1,1]]]);
var poly3 = turf.polygon([[[10,10],[10,15],[15,15],[15,10],[1

turf.booleanOverlap(poly1, poly2)
//=true
turf.booleanOverlap(poly2, poly3)
//=false
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

booleanParallel

npm install @turf/boolean-parallel

Boolean-Parallel returns True if each segment of line1 is parallel to the correspondent segment of line2

Arguments

Argument	Type	Description
line1	(Geometry|Feature <LineString>)	GeoJSON Feature or Geometry
line2	(Geometry|Feature <LineString>)	GeoJSON Feature or Geometry

Returns

boolean - true/false if the lines are parallel

Example

```
var line1 = turf.lineString([[0, 0], [0, 1]]);
var line2 = turf.lineString([[1, 0], [1, 1]]);

turf.booleanParallel(line1, line2);
//=true
```

booleanPointInP

npm install @turf/boolean-point-in-polygon

Takes a Point and a Polygon or MultiPolygon and determines if the point resides inside the polygon. The polygon can be convex or concave. The function accounts for holes.

Arguments

Argument	Type	Description
point	Coord	input point
polygon	Feature <(Polygon|MultiPolygon)>	input polygon or multipolygon

Argument	Type	Description
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
ignoreBoundary	boolean	false	True if polygon boundary should be ignored when determining if the point is inside the polygon otherwise false.

Returns

boolean - true if the Point is inside the Polygon; false if the Point is not inside the Polygon

Example

```
var pt = turf.point([-77, 44]);
var poly = turf.polygon([[
  [-81, 41],
  [-81, 47],
  [-72, 47],
  [-72, 41],
  [-81, 41]
]]);

turf.booleanPointInPolygon(pt, poly);
//= true
```

booleanPointOn

npm install @turf/boolean-point-on-line

Returns true if a point is on a line. Accepts a optional parameter to ignore the start and end

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

vertices of the linestring.

Arguments

Argument	Type	Description
pt	Coord	GeoJSON Point
line	Feature <LineString>	GeoJSON LineString
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
ignoreEndVertices	boolean	false	whether to ignore the start and end vertices.

Returns

boolean - true/false

Example

```
var pt = turf.point([0, 0]);
var line = turf.lineString([[-1, -1],[1, 1],[1.5, 2.2]]);
var isPointOnLine = turf.booleanPointOnLine(pt, line);
//=true
```

booleanWithin

npm install @turf/boolean-within

Boolean-within returns true if the first geometry is completely within the second geometry. The interiors of both geometries must intersect and, the interior and boundary of the primary (geometry a) must not intersect the exterior of the secondary (geometry b). Boolean-within returns the exact opposite result of the @turf/boolean-contains.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);
var point = turf.point([1, 2]);

turf.booleanWithin(point, line);
//=true
```

bearingToAzimut

npm install @turf/helpers

Converts any bearing angle from the north line direction (positive clockwise) and returns an angle between 0-360 degrees (positive clockwise), 0 being the north line

Arguments

Argument	Type	Description
bearing	number	angle, between -180 and +180 degrees

Returns

number - angle between 0 and 360 degrees

Note: bearingToAzimuth is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the bearingToAzimuth method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

convertArea

npm install @turf/helpers

Converts a area to the requested unit. Valid units: kilometers, kilometres, meters, metres, centimetres, millimeters, acres, miles, yards, feet, inches

Note: convertArea is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the convertArea method.

Arguments

Argument	Type	Description
area	number	to be converted
originalUnit	string	of the distance
finalUnit	string	returned unit

Returns

number - the converted distance

convertLength

npm install @turf/helpers

Converts a length to the requested unit. Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Note: convertLength is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the convertLength method.

Arguments

Argument	Type	Description
length	number	to be converted
originalUnit	string	of the length
finalUnit	string	returned unit

Returns

number - the converted length

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

degreesToRadiar

npm install @turf/helpers

Converts an angle in degrees to radians

Arguments

Argument	Type	Description
degrees	number	angle between 0 and 360 degrees

Returns

number - angle in radians

Note: degreesToRadians is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the degreesToRadians method.

lengthToRadians

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from a real-world unit into radians Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Arguments

Argument	Type	Description
distance	number	in real units
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - radians

Note: lengthToRadians is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the lengthToRadians method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lengthToDegrees

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from a real-world unit into degrees Valid units: miles, nauticalmiles, inches, yards, meters, metres, centimeters, kilometres, feet

Note: lengthToDegrees is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the lengthToDegrees method.

Arguments

Argument	Type	Description
distance	number	in real units
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - degrees

radiansToLength

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from radians to a more friendly unit. Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Note: radiansToLength is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the radiansToLength method.

Arguments

Argument	Type	Description
radians	number	in radians across the sphere
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - distance

radiansToDegree

npm install @turf/helpers

Converts an angle in radians to degrees

Arguments

Argument	Type	Description
radians	number	angle in radians

Returns

number - degrees between 0 and 360 degrees

Note: radiansToDegrees is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the radiansToDegrees method.

toMercator

npm install @turf/projection

Converts a WGS84 GeoJSON object into Mercator (EPSG:900913) projection

Arguments

Argument	Type	Description
geojson	(GeoJSON|Position)	WGS84 GeoJSON object
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description

Note: toMercator is part of the @turf/projection module.

To use it as a stand-alone module will need to import @turf/projection and call the toMercator method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - true/false

Example

```
var pt = turf.point([-71,41]);
var converted = turf.toMercator(pt);
```

toWgs84

npm install @turf/projection

Converts a Mercator (EPSG:900913) GeoJSON object into WGS84 projection

Note: toWgs84 is part of the @turf/projection module.

To use it as a stand-alone module will need to import @turf/projection and call the toWgs84 method.

Arguments

Argument	Type	Description
geojson	(GeoJSON|Position)	Mercator GeoJSON object
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - true/false

Example

```
var pt = turf.point([-7903683.846322424, 5012341.663847514]);
var converted = turf.toWgs84(pt);
```