Python

Command	Output	Command	Output
[dot] (pie / pi)	`.py`	lodge not	`!`
add comment	`#`	lodge or	`or`
breaker	`break`	long comment	`''''''`
class	`class`	long not	`not`
convert to character	`chr()`	make assertion	`assert`
convert to floating point	`float()`	open file	`open('filename', 'r') as f:`
convert to integer	`int()`	print to console	`print()`
convert to string	`str()`	read lines	`content = f.readlines()`
for each	`for in :`	return	`return`
for loop	`for i in range(0, ):`	self	`self`
from	`from`	shell iffae / LFA	`elif :`
function	`def`	shells	`else:`
global	`global`	sue iffae	`if`
identity is	`is`	sue shells	`else`
iffae	`if :`	try catch	`try: except Exception:`
import	`import`	value false	`False`
it are in	`in`	value not	`None`
jason	`json`	value true	`True`

length of	`len()`	while loop	`while :`
list comprehension	`[x for x in TOKEN if TOKEN]`	with	`with`
lodge and	`and`		

Javascript

Command	Output	Command	Output
Let	`let`	lodge and	`&&`
add comment	`//`	lodge not	`!`
anon funk	`() => { }`	lodge or	| |
breaker	`break;`	long comment	`/**/`
case of	`case :`	new new	`new`
catch	`catch(e) { }`	print to console	`console.log()`
const	`const`	push	`push`
continue	`continue`	return	`return`
convert to floating point	`parseFloat()`	self	`self`
convert to integer	`parseInt()`	shell iffae	`else if ()`
convert to string	`""+`	shells	`else { }`
default	`default:`	switch	`switch() { }`
do loop	`do { }`	this	`this`
document	`document`	throw	`throw`
for each	`for (TOKEN in TOKEN)`	timeout	`setTimeout()`

for loop	`for (var i=0; i<TOKEN; i++)`	timer	`setInterval()`
function	`function TOKEN() { };`	try	`try { }`
has own property	`hasOwnProperty()`	value false	`false`
iffae	`if () { }`	value not	`null`
index of	`indexOf()`	value true	`true`
inner HTML	`innerHTML`	var	`var`
instance of	`instanceof`	while loop	`while ()`
length	`length`		

Java

Command	Output	Command	Output
add comment	`//`	iterate and remove	`for (Iterator<TOKEN> iterator = TOKEN.iterator(); iterator.hasNext();) { String string = iterator.next(); if (CONDITION) { iterator.remove(); } }`
array list	`ArrayList`	lodge and	`&&`
arrow	`->`	lodge not	`!`
big double	`Double`	lodge or	`| |`
big integer	`Integer`	long comment	`/**/`
boolean	`boolean`	new new	`new`
breaker	`break;`	print to console	`java.lang.System.out.println()`
case of	`case :`	private	`private`

cast to double	`(double)()`	public	`public`

C++

Command	Output	Command	Output
([global] scope / name)	`::`	integer	`int`
(pointer / D reference)	`*`	lodge and	`&&`
(reference to / address of)	`&`	lodge not	`!`
Vic	`vector`	lodge or	`||`
add comment	`//`	long comment	`/**/`
array	`Brackets`	member	`->`
big integer	`Integer`	new new	`new`
breaker	`break;`	print to console	`cout <<`
case of	`case :`	private	`private`
character	`char`	public	`public`
class	`class TOKEN{}`	pushback	`push_back`
constant	`const`	return	`return`
convert to floating point	`(double)`	shells	`else{}`
convert to integer	`(int)`	standard	`std`
convert to string	`std::to_string()`	static	`static`

default	`default:`	static cast double	`static_cast<double>()`
do loop	`do {}`	static cast integer	`static_cast<int>()`
double	`double`	string	`string`
final	`final`	switch	`switch(){ case : break; default: break;}`
for each	`for_each (TOKEN, TOKEN, TOKEN);`	ternary	`()?;`
for loop	`for (int i=0; i<TOKEN; i++)`	value false	`false`
function	`TOKEN TOKEN(){}`	value not	`null`
iffae	`if(){}`	value true	`true`
import	`#include`	while loop	`while ()`

Go

Command	Output		Command	Output
iffae	`if  {` `}`		shells	`else {` `}`
switch	`switch  {` `}`		case of	`case :`
breaker	`break`		default	`default:`
while loop	`for  {` `}`		for loop	`for i := 0; i<; i++ {` `}`
for each	`for i := range  {` `}`		convert to integer	`strconv.Atoi()`
convert to string	`strconv.Itoa()`		lodge and	`&&`
lodge or	| |		lodge not	`!`

print to console	`fmt.Println()`		import	`import ( )`
function	`func`		class	`type  struct { }`
add comment	`//`		long comment	`/**/`
value not	`nil`		return	`return`
value true	`true`		value false	`false`
(inter / integer)	`|  | boolean |`			
string	`string`		assign	`:=`
(function / funk) main	`func main() { }`		make map	`make(map[])`
package	`package`			``

R

Command	Output	Command	Output
`<function>`	`<function>()`	lodge not	`!`
NA	`NA`	lodge or	| |
add comment	`#`	print to console	`print()`
assign	`<-`	return	`return()`
breaker	`break`	see as vee	`csv`
contained in	`%in%`	shells	`else`
dot (our/are)	`.R`	slurp / chain	`%>%`
for each	`for ( in ):`	tell (slurp / chain)	`{end of line} %>% {newline}`
for loop	`for (i in 1:)`	tell add	`{end of line} + {newline}`

function	`function()`	tidy verse	`tidyverse`
`graph <ggfun>`	`<ggfun>()`	value false	`FALSE`
iffae	`if ()`	value not	`NULL`
import	`library()`	value true	`TRUE`
lodge and	`&&`	while loop	`while ()`

SQL

Command	Output	Command	Output
alias as	`AS`	it are in	`IN`
ascending	`ASC`	join	`JOIN`
between	`BETWEEN`	left join	`LEFT JOIN`
delete	`DELETE`	like	`LIKE '%'`
descending	`DESC`	lodge and	`AND`
equals / equal to	`=`	lodge or	`OR`
from	`FROM`	not equals / not equal to	`<>`
full join	`FULL JOIN`	on columns	`ON`
fun average	`AVG()`	order by	`ORDER BY`
fun count	`COUNT()`	over partition by	`OVER (PARTITION BY )`
fun max	`MAX()`	right join	`RIGHT JOIN`
fun min	`MIN()`	select	`SELECT`
group by	`GROUP BY`	select (all / every)	`SELECT *`
inner join	`INNER JOIN`	union	`UNION`
insert into	`INSERT INTO`	update	`UPDATE TOKEN SET`

is not null	`IS NOT NULL`	using	`USING ()`
is null	`IS NULL`	where	`WHERE`

Bash

Command	Output	Command	Output
add comment	`#`	lodge and	`&&`
breaker	`break`	lodge not	`!`
case of	`TOKEN) ;;`	lodge or	`||`
continue	`continue`	print to console	`echo`
default	`*) ;;`	push	`TOKEN+=()`
do loop	`until [ ]; do`	return	`return`
end switch	`esac`	she bang	`#!/bin/bash`
for each	`for TOKEN in TOKEN; do`	shell iffae	`elif [[ ]];`
for loop	`for (( i=0; i<=TOKEN; i++ )); do`	shells	`else`
function	`TOKEN(){}`	sue iffae	`[[ ]]`
iffae	`if [[ ]];`	switch	`case TOKEN in`
import	`. /path/to/functions`	value false	`false`
key do	`do`	value not	`-z "$var"`
key done	`done`	value true	`true`
key fee	`fi`	while loop	`while [ ]; do`
length of	`${#TOKEN[@]}`		