Rosetta Stone for Scripting Languages

Language	sh	perl	python
Website		www.perl.org	www.python.org
Author	Steve Bourne	Larry Wall *et al*	Guido van Rossum
License		GPL / Perl Artistic License	CNRI Open Source GPL-Compatible license
Single Line Invocation	`sh -c "script"`	`perl -e 'script'`	`python -c script`
Script File Invocation	`sh scriptfilename`	`perl scriptfilename`	`python scriptfilename`
Whitespace	Mostly not significant	Not significant	Significant: indentation indicates blocks
Statement Separator	; or newline	;	Newline
Block		{ }	(Indicated by increased indentation)
Comment	#	#	#
Variable	`var` when setting; `${var}` or `$var` otherwise	`$var`	`var`
Array		`$array[n]` or `@array` for whole thing; `$array[0]` for first element	`(1,2,3)` tuple (immutable), `[1,2,3]` list (mutable)
Array Size		`$#array + 1`	`len sequence`
Hash		`$hash{key}` or `%hash` for whole thing	`{ key:value,..., key:value}`
Hash Iterate		`foreach $key (keys %hash)`	`for key,value in hash.items()`

String	`"quoted string"` (with variable/command substitution),
	`'quoted string'` (without variable/command substitution)
		`"quoted string"` (with variable/command substitution),
		`'quoted string'` (without variable/command substitution)
			`'Quoted string'`
			`"Quoted string"`
			`'''Quoted string including line breaks'''`

Command line arguments	`$0`,`$1`,`$n` where *n* is `$#` (and only up to `$9`; after that, use `shift`)	`@ARGV`	`sys.argv`
Last Result		`$_`	_ (but only for explicit expressions in an interactive session)
Assignment	`var=value`	`$var = value;`	`var = value`
Equality (string)	=	eq	==
Equality (numeric)	-eq	==	==
Logical And	-a	&&	and
Logical Or	-o	||	or
Logical Not	!	!	not

Regexp Search		*/regexp/*	`import re`
			`reobj = re.compile(regexp)`
			`reobj.search(string)` (for repeated searches)
			or (for one-time matching)
			`re.search(regexp, string)`

Regexp Replace		*s/regexp/replacement/*	`result = reobj.sub(replacement, string)`
			or
			`result = re.sub(regexp, replacement, string)`

Conditional	`if condition`
	`then block`
	`elif condition`
	`then block`
	`else block`
	`fi`
		`if (test) {block}`
		`elsif (test) {block}`
		`else {block}`
			`if condition:`
			`  block`
			`elif condition:`
			`  block`
			`else:`
			`  block`

Iteration	`for item in list`
	`do`
	`  block`
	`done`
		`for ($i=1; $i<10; $i++) {block}`
		`for (10,9,8,7,6,5,4,3,2,1) {block}`
		`for (1..15) {block}`
		`foreach $element (@array) {block}`
			`for x in iterable:`
			`  block`

Loop	`while condition`
	`do`
	`  block`
	`done`
		`while (test) {block}`	`while condition:`
			`  block`

Next Loop Iteration	continue	next	continue
Exit Loop	break	last	break

Switch	`case value in`
	`  pattern1) commands;;`
	`  pattern2) commands;;`
	`esac`		

Function Definition	`fnname () { commands; }`	`sub fnname {block}` with arguments in `@_`	`def fnname(param1, param2,..):` `  block`
Function Call	fnname	`&fnname(arg1, arg2)`	`fn(arg1, arg2, ..)`
Function Parameters pass by	Value	Value or reference (with @@@)	Value
Print to stdout	print	`print STDOUT "stuff to print\n",` `printf STDOUT format ...`	print
Open File for Read		`open(FILEHANDLE, "filename")`	`open(filename, mode='r')`
Open File for Write		`open(FILEHANDLE, ">filename")`	`open(filename, mode='w')`
Read from File	`read` (only from `stdin`)	`<FILEHANDLE>`	`f.read()`
Write to File		`print FILEHANDLE "stuff to print\n"`	`f.write(string)`
Close File		`close(FILEHANDLE)`	`f.close()`

Test File	`test -r` readable,
	`test -f` is plain file,
	`test -d` is directory;
	`[ condition ]` is alternative form
		`-r` readable,
		`-e` exists,
		`-f` is plain file,
		`-d` is directory
			`os.path.exists(filename)`
			`os.path.isfile(filename)`
			`os.path.isdir(filename)`

Delete File	`rm filename`	`unlink("filename")`	`os.remove(filename)`
Rename File	`mv oldfilename newfilename`	`rename("oldname", "newname")`	`os.rename(oldname, newname)`

File Info		`($dev, $ino, $mode, $nlink, $uid,`
		`$gid, $rdev, $size, $atime, $mtime, $ctime,`
		`$blksize, $blocks) = stat(FILEHANDLE)`
			`os.stat(filename)`

Output Formatting		C-like `printf` formats	
String Concatenation		`"stringa" . "stringb"`	`string1+string2`
Substring	`${stringvar:offset:length}` (bash only)	`substr(string, offset [,length])`	`string[offset:endoffset]`
String Tokenization		`split /separator/ expression`	`string.split(separator)`
Language	**sh**	**perl**	**python**