Module Management

`dir()`	list all objects in the programs namespace
`import module`	import *module* into its own namespace
`import module as alt`	import *module* into the namespace *alt*
`from module import func`	import the function *func* from *module* into the programs namespace
`dir(module)`	list all objects in the namespace *module*
`reload module`	reinitialise *module*

Useful Modules (General)

`sys`	interacting with the system command line
`os`	interacting with the operating system
`fileinput`	simple input file processing
`argparse`	easily manage command line options and arguments
`random`	tools for generating pseudo-random data
`collections`	an expanded set of helpful data structures
`glob`	working with wildcards
`re`	regular expression searching/matching
`datetime`	time and date handling/processing
`urllib / urllib2`	working with webpages

Useful Modules (Data Handling/Plotting/Biology)

`numpy`	tools for working with numerical data
`scipy`	scientific computing
`pandas`	improved data handling
`sklearn`	extensive machine learning module
`blaze`	working with very large datasets
`matplotlib`	powerful plotting library
`bokeh`	simple interactive plotting
`seaborn`	easy plotting for statistical data
`ggplot`	plotting similar to ggplot2 in R
`Bio`	Biopython (see biopython.org)

Statements - Simple

`assert`	Used in debugging to check expressions
`pass`	Do nothing
`del`	Delete name or value
`print`	Output to STDOUT
`return`	Used in functions to return a value
`yield`	Used in generator functions
`break`	Terminate inner-most enclosing loop
`continue`	Skip to next iteration of enclosing loop
`raise`	Raise an exception
`global`	Mark variable as globally available
`exec`	Execute code contained in a string

Statements - Compound

`if:elif:else:`	Conditional execution of code blocks
`while:else:`	Repeat code block while expression is true
`for in:else:`	loop through entries in a sequence
`try…except… finally`	exception handling of risky code
`with`	define context for a block of code
`def`	define a new function object
`class`	define a new class

Logical Tests

`==`	is equal to
`!= , <>`	is not equal to
`and`	and
`or`	or
`not`	not
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`is`	true if the operands refer to the same object
`is not`	opposite of the above
`in`	true if first operand contained within the second operand (a sequence)
`not in`	opposite of the above

Python Programming Language Cheat Sheet

Standard Operators

`+`	Addition (or string concatenation)
`-`	Subtraction (or unary minus)
`*`	Multiplication (or string repetition)
`/`	Division
`%`	Modulus (remainder)
`//`	Floor division
`**`	Exponention

Assignment Operators

`x = y`	standard assignment
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`
`x //= y`	`x = x //y`
`x <<= y`	`x = x << y`
`x >>= y`	`x = x >> y`
`x &= y`	`x = x & y`
`x |= y`	`x = x | y`
`x ^= y`	`x = x ^ y`

Bitwise Operators

`&`	Bitwise AND
`|`	Bitwise OR
`^`	Bitwise XOR
`~`	Bitwise NOT
`x << n`	Shift bits of *x* left by *n* bits
`x >> n`	Shift bits of *x* rightt by *n* bits

Common Datatype Functions

`len(item)`	returns the length of *item*
`max(item)`	returns the maximum value contained in a list or tuple
`min(item)`	returns the minimum value contained in a list or tuple
`any()`	returns true if any element is true
`all()`	returns true if all elements are true

Dictionary Methods

`clear()`	remove all entries in the dictionary
`copy()`	create a 'shallow' copy of dictionary
`get(key [,default])`	returns value associated with *key*, or default if *key* is not in the dictionary
`has_key(key)`	returns True if *key* exists in dictionary
`items()`	returns list of key-value pairs as tuples
`keys()`	returns a list of the keys in the dictionary
`pop(key [,default])`	removes entry for *key* & returns associated value or *default* if *key* is not in dictionary
`setdefault( key[,default])`	returns value associated with *key*, or creates *key-default* entry & returns *default* if *key* is not in dictionary
`update(dct2)`	adds entries of *dct2* into dictionary
`values()`	returns all values in a list

Indices, Slices, Ranges

`a[N]`	returns Nth item in sequence *a*
`a[-1]`	returns the last element in *a*
`a[-4]`	returns the 4th-from-last element of *a*
`a[N:]`	returns from Nth element to the end
`a[:N]`	returns from first to N-1th element
`a[N:M]`	returns from Nth to M-1th element
`range(N)`	returns list of integers from 0 - *N*-1
`range(M,N)`	returns list of integers from *M* - *N*-1
`range(M,N,L)`	returns list of integers from *M* - *N*-1 in steps of size *L*

List Methods

`append(item)`	adds *item* to the end of the list
`count(value)`	returns count of occurrences of *value* in list
`extend(lst2)`	adds elements in *lst2* to end of list
`index(value)`	returns index of 1st occurrence of *value* in list
`insert(i, item)`	inserts *item* into list at position *i*
`pop(i)`	removes and returns ith value
`remove(item)`	removes first occurence of *item*
`reverse()`	reverses the order of the list
`sort()`	sorts list elements by value

String Methods (Formatting)

`capitalize()`	capitalise the first letter of every word
`center(width[, fillchar])`	center-align string in a field of *width* characters, filled by *fillchar* or spaces by default
`lower()`	convert all letters to lowercase
`rstrip()`	removes whitespace from the righthand side of the string
`strip()`	removes whitespace from both sides of the string
`swapcase()`	swaps uppercase to lower and *vice versa*
`upper()`	converts all letters to uppercase

String Methods (Searching)

`count(sub[, start[,end]])`	count the number of occurrences of *sub* in string, between *start* and *end* positions
`find(sub[, start[,end]])`	returns position of *sub* between *start* and *end*, or -1 if not found
`index(sub[, start[,end]])`	returns position of *sub* between *start* and *end*, raises error if not found
`replace(old, new)`	replace all instances of *old* in string with *new*
`split(sep[, maxitems])`	returns list of substrings split up by *sep*, optionally into *maxitems* items
`join(seq)`	returns a string of all entries in seq concatenated, with string between entries

String `.format()` Placeholder Codes

`{}`	inserts the value of the next method argument
`{N}`	inserts value of the Nth method argument
`{N:d}`	inserts Nth value as a decimal integer
`{N:.4f}`	inserts Nth value as a floating point to 4 decimal places
`{:,}`	splits large numbers into thousands by commas
`{:.2%}`	inserts value represented as a percentage, to two decimal places
`{N:x<10s}`	inserts a string, left-aligned in a field of width 10, with any remaining space filled with 'x' characters

String Methods (Testing)

`startswith( sub)`	true if string starts with substring *sub*
`endswith(sub)`	true if string ends with substring *sub*
`isalnum()`	true if all characters are alphanumeric
`isalpha()`	true if all characters are alphabetic
`isdigit()`	true if all characters are numeric
`islower()`	true if all characters are lowercase
`isspace()`	true if all characters are whitespace
`istitle()`	true if string is titlecase (all words are capitalised)
`isupper()`	true if all characters are uppercase

File Methods

`close()`	close connection to file
`readlines()`	read contents one line at a time
`readline()`	read the next line in the file
`read(size)`	read *size* bytes from the file
`fileno()`	returns the file number from the OS
`truncate(size)`	delete file contents after *size*
`write(string)`	write *string* to file
`writelines( list)`	write strings in *list* as lines to file

originally compiled by P D Ashton, adapted and updated by T Hodges in 2015