Help on class str: (Assume S is a string)

S.capitalize() -> str
Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

S.casefold() -> str
Return a version of S suitable for caseless comparisons.

S.center(width[, fillchar]) -> str
Return S centered in a string of length width. Padding is done using the specified fill character (default is a space)

S.count(sub[, start[, end]]) -> int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

S.endswith(suffix[, start[, end]]) -> bool
Return True if S ends with the specified suffix, False otherwise.
With optional start, test S beginning at that position.
With optional end, stop comparing S at that position.
suffix can also be a tuple of strings to try.

S.expandtabs(tabsize=8) -> str
Return a copy of S where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

S.find(sub[, start[, end]]) -> int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.
Return -1 on failure.

S.format(*args, **kwargs) -> str
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

S.index(sub[, start[, end]]) -> int
Like S.find() but raise ValueError when the substring is not found.

S.isalnum() -> bool
Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

S.isalpha() -> bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

S.isdecimal() -> bool
Return True if there are only decimal characters in S, False otherwise.

S.isdigit() -> bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

S.islower() -> bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

S.isnumeric() -> bool
Return True if there are only numeric characters in S, False otherwise.

S.isprintable() -> bool
Return True if all characters in S are considered printable in repr() or S is empty, False otherwise.

S.isspace() -> bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

S.istitle() -> bool
Return True if S is a titlecased string and there is at least one character in S, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones.
Return False otherwise.

S.isupper() -> bool
Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

S.join(iterable) -> str
Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

S.ljust(width[, fillchar]) -> str
Return S left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

S.lower() -> str
Return a copy of the string S converted to lowercase.

S.lstrip([chars]) -> str
Return a copy of the string S with leading whitespace removed.
If chars is given and not None, remove characters in chars instead.

S.partition(sep) -> (head, sep, tail)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

S.replace(old, new[, count]) -> str
Return a copy of S with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

S.rfind(sub[, start[, end]]) -> int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.
Return -1 on failure.

S.rindex(sub[, start[, end]]) -> int
Like S.rfind() but raise ValueError when the substring is not found.

S.rjust(width[, fillchar]) -> str
Return S right-justified in a string of length width. Padding is done using the specified fill character (default is a space).

S.rpartition(sep) -> (head, sep, tail)
Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

Help on class str continued: (Assume S is a string)

S.rsplit(sep=None, maxsplit=-1) -> list of strings
Return a list of the words in S, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.

S.rstrip([chars]) -> str
Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead.

S.split(sep=None, maxsplit=-1) -> list of strings
Return a list of the words in S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

S.splitlines([keepends]) -> list of strings
Return a list of the lines in S, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

S.startswith(prefix[, start[, end]]) -> bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

S.strip([chars]) -> str
Return a copy of the string S with leading and trailing whitespace removed. If chars is given and not None, remove characters in chars instead.

S.swapcase() -> str
Return a copy of S with uppercase characters converted to lowercase and vice versa.

S.title() -> str
Return a titlecased version of S, i.e. words start with title case characters, all remaining cased characters have lower case.

S.upper() -> str
Return a copy of S converted to uppercase.

S.zfill(width) -> str
Pad a numeric string S with zeros on the left, to fill a field of the specified width. The string S is never truncated.

Writing textfiles:
open(filename,'w') -> file
Returns a file for writing
If the file already exists, it is overwritten.

F.write(string) -> integer
Writes the string to the file.
Returns the number of characters written.

Other Important String Functionality:

chr(integer) -> str
Returns a one-character string with ASCII code integer. integer must be in the range [0..255]

len(str) -> integer
Returns the number of characters in the string.

ord(str) -> integer
Returns the ASCII value of a one-character string. str must be a string of a single character.

str(object) -> str
Converts an object (like a number) into a string.

Accessing portions of a string can be done with slicers:
Given a string S,

- S[n] will return the $(n+1)^{st}$ character of the string.
- S[n:p] will return the substring starting at the $(n+1)^{st}$ character (character S[n]) and ending at the p^{th} character (character S[p–1]). Note: p must be greater than n.
- S[n:] will return the substring starting at the $(n+1)^{st}$ character (character S[n]) and going to the end of the string.
- S[:p] will return the substring starting at the beginning of the string and ending at the p^{th} character (character S[p–1]).
- S[n:p:q] will return the substring starting at the $(n+1)^{st}$ character (character S[n]) and ending at the p^{th} character (character S[p–1]), counting every q^{th} character. Note: p must be greater than n, unless q is negative, in which case, the string will be traced backwards.
- Note that if n and p are negative, the characters will count from the end, where the last character is S[–1], the second-to-last character is S[-2], etc.

Reading textfiles:
open(filename) or open(filename,'r') -> file
Returns a file for reading

F.read() -> str
Returns the entire file as a string.

F.readline() -> str
Returns a line from the text file, including the terminating newline character.
Returns an empty string if the end of file has been reached.

F.readlines() -> list of strings
Returns a list of strings, where each item in the list is one line from the file.