Some useful Python functions:
- `abs(x)` returns the absolute value of x.
- `m%n` returns the remainder when integer m is divided by positive integer n.
- `m/n` corresponds to integer division when m and n are both integers (that is, the quotient).
- `min(...)` returns the minimum value among its arguments (which may be numbers, strings, or a list).
- `max(...)` returns the maximum value among its arguments (which may be numbers, strings, or a list).
- `print(x,end=s)` prints x on a current line, followed by s (or followed by '\n' if end not specified).
- `check.expect(comment,value1,value2)` prints a summary message indicating if the test passed (if `value1 == value2`) or failed.
- `check.within(comment,value1,value2,tol)` prints a summary message indicating if the test passed (if `abs(value1 - value2) ≤ tol`) or failed.
- `check.set_input(x1,x2,…,xn)` sets up program to use parameters when `input` is called by a tested function
- `check.set_screen(s)` prints s as expected output when a test is run.
- `check.set_print_exact(x1,x2,…,xn)` checks parameters against printed output when test is run.
- `type(x)` returns the type of value x. For example, you may check if x is of type Bool with `type(x) == type(True)`.
- Type conversion operations include: `str(x)`, `int(x)`, `float(x)`, `list(x)`.

String functions and methods in Python:
- `len(s)` returns the number of characters in s.
- `s[a:b]` returns a string containing the characters at positions a, a+1, ... b-1 for 0≤a≤b≤len(s). There is no error if b > len(s).
- `s[a:b:c]` returns a string containing the characters at positions a, a+c, a+2c, ... The last character in the new string comes before position b in s.
- `s in t` returns True if string s occurs as a substring in t, and False otherwise.
- `s + t` returns a new string containing the characters of string s followed by the characters of string t.
- `input(p)` returns a string entered by keyboard input after the prompt p is displayed. Returned string does not include newline character.
- `s.count(c)` returns the number of times string c occurs in string s (could be 0).
- `s.format(v0,v1,...)` returns a string like s, except that v0 replaces {0}, v1 replaces {1}, etc.
- `s.find(t)` or `s.find(t, pos)` returns the index of the first occurrence of t in s (returns -1 if t is not a substring of s) starting from position pos if given.
- `s.isalnum()` returns True if s is nonempty and all characters are alphabetical (letters) or numeric (digits), and False if the string is empty or is nonempty and at least one character is not alphabetical or numeric.
- `s.isdigit()` returns True if all characters in s are digits ('0',...,'9'), and False otherwise. Returns False for the empty string.
- `s.islower()` returns True if all characters in s are lowercase, and False otherwise. Returns False for the empty string.
- `s.isupper()` returns True if all characters in s are uppercase, and False otherwise. Returns False for the empty string.
- `s.join(L)`, where L is a (listof Str), returns the string L[0]+s+L[1]+s+...+s+L[-1].
- `s.lower()` returns a string like s, except all uppercase characters are replace by lowercase versions.
- `s.replace(a,b)` returns a new string like s, except that all occurrences of a are replaced with b.
- `s.split()` returns a list of strings from s, by dividing s at whitespace. If s has value "my dog has fleas.\n", then s.split() returns ["my","dog","has","fleas."].
- `s.startswith(t)` returns True if string s begins with the string t, and False otherwise.
- `s.strip()` returns a string like s, but leading and trailing whitespace (including newline characters) are removed.
- `s.upper()` returns a string like s, except all lowercase characters are replaced by uppercase versions.
List functions and methods in Python:

- `len(L)` returns the number of values in `L`.
- `sum(L)` returns the sum of all entries in `L` (must be numbers).
- `L[a:b]` returns the list `[L[a], L[a+1], ..., L[b-1]]` for `0 \leq a \leq b \leq \text{len}(L)`. There is no error if `b > \text{len}(L)`.
- `L[a:b:c]` returns the list `[L[a], L[a+c], L[a+2*c], ...]`. The last item in the new list comes before position `b` in `L`.
- `list(map(func,lst))` returns the list that results from applying `func` to each element of `lst` (also works if `lst` is a string).
- `list(filter(func,lst))` returns the list of all elements of `lst` for which `func` returns `True` (also works if `lst` is a string).
- `x in L` returns `True` if `x` is an element of `L`, and `False` otherwise.
- `L+M` returns a new list containing the elements of the list `L` followed by the elements of the list `M`.
- `L.extend(M)` returns `None` and mutates the list `L` by adding the elements of list `M` to the end of list `L`.
- `L.append(x)` returns `None` and mutates the list `L` by placing the value `x` at the end of the list `L`.
- `L.index(x)` returns the smallest index `j` such that `L[j] = x` if `x` is in `L`, and results in an error if `x` is not in `L`.
- `L.insert(p,x)` returns `None` and mutates the list `L` by inserting `x` into position `p`, and keeping other values in `L` in the same relative positions.
- `L.remove(x)` returns `None` and mutates the list `L` by removing the first occurrence of the value `x`, and results in an error if `x` is not in `L`.
- `L.pop(k)` returns `L[k]` and mutates the list `L` by removing the value at position `k`, and results in an error if `k` is not a valid list position.
- `L.sort()` returns `None` and mutates the list `L` by sorting it into increasing order.
- `L.reverse()` returns `None` and mutates the list `L` by reversing the order of the elements.