Some Useful Python functions and operations:

- `m%n` returns the remainder when the non-negative integer `m` is divided by the positive integer `n`.
- `m//n` corresponds to integer division when `m` and `n` are both integers (i.e., the quotient).
- `max(...)` returns the maximum value among its arguments (which may be strings, numbers or a list)
- `min(...)` returns the minimum value among its arguments (which may be strings, numbers or a list)
- `abs(x)` returns the absolute value of the number `x`
- `list(range(b))` returns `[0,1,2,...,b-1]`.
- `list(range(a,b))` returns `[a,a+1,...,b-1]`.
- `list(range(a,b,-1))` returns `[a,a-1,...,b+1]`.
- `list(map(func,C))` returns the list that results from applying `func` to each element of list or string `C`.
- `list(filter(func,C))` returns the list of all elements of list or string `C` for which `func` returns `True`.
- `check.expect(comment, value1, value2)` prints a message indicating if the test passed or not.
- `check.within(comment, value1, value2, tol)` prints a message indicating if the test passed or not.
- `check.set_input(x1,...,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,...,xn)` checks parameters against printed output when test is run.
- `check.set_file_exact(file1,file2)` prints a summary of the differences in file1 and file2

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)`. `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`.
- `s * n` returns a new string containing `n` copies of `s`.
- `input(p)` returns a string entered by keyboard input after the prompt `p` is displayed. Produced 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.index(t)` returns the index of the first occurrence of `t` in `s` (returns error if `t` is not a substring of `s`)
- `s.isalpha()` returns `True` if `s` is nonempty and all characters are alphabetical (letters), and `False` if the string is empty or it is nonempty and at least one character is not alphabetical.
- `s.isdigit()` returns `True` if all characters in `s` are digits ('0','1',...,'9'), and `False` otherwise. Returns `False` for the empty string.
- `s.islower()` returns `True` if all characters in `s` are lowercase, and `False` otherwise (including empty string).
- `s.isupper()` returns `True` if all characters in `s` are uppercase, and `False` otherwise (including empty string).
- `s.join(L)`, where `L` is a list of 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.split()` returns a list of strings from `s`, by dividing `s` at whitespace.
- `s.split(sep)` returns a list of strings from `s`, by dividing `s` at `sep`.
- `s.strip()` returns a string like `s`, but with leading and trailing whitespace removed.
- `s.upper()` returns a string like `s`, except all lowercase characters are replace by uppercase versions

Dictionary functions and methods in Python:

- `len(D)` returns the number of keys in the dictionary `D`.
- `k in D` returns `True` if `k` is a key in the dictionary `D`, and `False` otherwise.
- `list(D.keys())` returns the list of keys in the dictionary `D`.


• `list(D.values())` returns the list of associated values in the dictionary D.
• `D.pop(key)` removes key from dictionary D and returns its value if key is in the dictionary, otherwise a `KeyError` is raised.

**List functions and methods in Python:**
• `len(L)` returns the number of values in `L`.
• `L[a:b]` returns the list `[L[a], L[a+1], ..., L[b-1]]` for `0<=a<b<=len(s)`. There is no error if `b > 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`.
• `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`.
• `[x] * n` returns a new list containing `n` copies of `x`.
• `sum(L)` returns the sum of all values in the list of numbers `L`.
• `L.count(x)` returns the number of times that `x` occurs in `L`.
• `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 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.pop(k)` returns `L[k]` and mutates the list `L` by removing the value at position `k` (note: error if `L` is empty).
• `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.sort()` returns `None` and mutates the list `L` by sorting it into increasing order.

**File functions and methods in Python:**
• `open(fn, m)` returns a file object and opens the file named `fn` for reading if `m='r'`, and for writing if `m='w'`.
• `f.readline()` returns the string, including the newline character (if it is in the file), that is the next line of the file `f`, opened for reading. At the end of the file, it returns the empty string.
• `f.readlines()` returns the list of strings, each ending with a newline character if it is present in the file, which can be obtained from the file `f`, opened for reading. At the end of the file, it returns `[]`.
• `f.write(s)` adds the string `S` to the end of the file `f`, opened for writing. It does not add a newline character to `f` unless that character is in `s`. Returns the number of characters written to `f`.
• `f.writelines(L)` adds each string in `L` to the end of the file `f`, opened for writing. It does not add any newline characters to `f` unless they are in the strings in `L`.
• `f.close()` closes the file `f`.

**Useful recurrence relations for analyzing runtime of recursive code:**

- \[ T(n) = O(1) + T(n-1) \text{ is } O(n) \]
- \[ T(n) = O(n) + T(n-1) \text{ is } O(n^2) \]
- \[ T(n) = O(1) + T(n/2) \text{ is } O(\log n) \]
- \[ T(n) = O(n) + 2T(n/2) \text{ is } O(n \log n) \]
- \[ T(n) = O(n) + T(n/2) \text{ is } O(n) \]

- \[ T(n) = O(1) + 2T(n/2) \text{ is } O(n) \]
- \[ T(n) = O(1) + T(n-1) + T(n-2) \text{ is } O(2^n) \]
- \[ T(n) = O(n) + T(n-1) + T(n-2) \text{ is } O(2^n) \]
- \[ T(n) = O(1) + 2T(n/2) \text{ is } O(2^n) \]
- \[ T(n) = O(n) + 2T(n/2) \text{ is } O(2^n) \]

**Assumed run-times for built-in Python functions**

- `len(x)` is `O(1)`, for any list, string, or dictionary `x`.
- `list(range(n))` is `O(n)`.
- `list operations: in, count, sum, index, remove, pop` are `O(n)` for lists of length `n`, while `append` is `O(1)`.
- `string operation such as in, islower, lower, index, etc.,` are all `O(n)`.
- `dictionary operations: in, key retrieval, adding key-value pairs, updating` are `O(1)`, keys and values are `O(n)`.