Question 1. [1 MARK]

Convert the following for loop into a while loop.
`string_list` is a non-zero length list that contains only strings.
`lower_list` is an initially empty list.

```
for elt in string_list:
    if elt.is_upper():
        lower_list.append(elt.lower())
```

Solution:

```
i = 0

while i < len(string_list):
    if string_list[i].is_upper():
        lower_list.append(string_list[i].lower())
    i += 1
```

Question 2. [2 MARKS]

Convert the following for loop into a while loop.
`end` is an integer.
`in_list` is a nested list that contains at least end lists.

```
ret_list = []
for i in range(2, end, 4):
    ret_list.append(i)
    if '*' in in_list[i]:
        ret_list[-1] = ret_list[-1]**2
```

Solution:

```
ret_list = []
i = 2

while i < end:
    ret_list.append(i)
    if '*' in in_list[i]:
        ret_list[-1] = ret_list[-1]**2
    i += 4
```

Question 3. [3 MARKS]

Convert the following while loop into a for loop.
You code should be written so that the return statement does not need to be altered.
You are allowed to change the code before the while loop proper.
`string_list` is a list of strings than contains at least 15 instances of the string '***'.

```
j = 0
while i < 15:
    if string_list[j] == '***':
        i += 1
    j += 1

return j
```

Solution:

```
j = 0
counter = 0
for i in range(len(string_list)):
    if string_list[i] == '***':
        counter += 1
    if counter == 15:
        j = i + 1 #Note that in the orginal code, j refers to the index of the
                  #first element after the 15th '***'

return j
```

Question 4. [4 MARKS]

Rewrite this block of code so that there are no indents.
`x` is an integer.

```
if (x % 2) == 0:
    if (x < 50):
        return True
    else:
        return False
else:
    if (x < 50):
        return False
    else:
        return False
```

Solution:

```
return ((x % 2) == 0) and (x < 50)
```

Question 5. [5 MARKS]

Write a function called `line_to_list` so that the code with * comments is replaced by a single line that includes a call to `line_to_list`.
`input_file` is a `file`.

```
nested_list = []
for line in input_file:
    line_list = [] #*
    for char in line: #*
        line_list.append(char) #*
    nested_list.append(line_list) #*
```

Solution:

```
def line_to_list(in_str):
    '''(str -> list)
    Return a list where element i is the corresponding character of in_str.'''
    ret_list = []
    for char in in_str:
        ret_list.append(a)
    return ret_list

nested_list = []
for line in input_file:
    nested_list.append(line_to_list(line))
```

Short Python function/method descriptions:

```
__builtins__:
  len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
  max(L) -> value
    Return the largest value in L.
  cmp(x, y) -> integer
    Return negative if x<y, zero if x==y, positive if x>y.
  min(L) -> value
    Return the smallest value in L.
  open(name[, mode]) -> file object
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with start and ending with
    stop - 1 with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.
  raw_input([prompt]) -> string
    Read a string from standard input.  The trailing newline is stripped.
cPickle:
  dump(obj, file)
    Write an object in pickle format to the given file.
  load(file) --> object
    Load a pickle from the given file
dict:
  D[k] --> value
    Return the value associated with the key k in D.
  k in d --> boolean
    Return True if k is a key in D and False otherwise.
  D.get(k) -> value
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
  D.items() -> list of (key, value) pairs
    Return the (key, value) pairs of D, as 2-tuples.
file (also called a "reader"):
  F.close()
    Close the file.
  F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF (End
    of File) is reached.
  F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete
    line may be returned then).  Return an empty string at EOF.
float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
int:
  int(x) -> integer
    Convert a string or number to an integer, if possible.  A floating point
    argument will be truncated towards zero.
list:
```

```
  x in L --> boolean
    Return True if x is in L and False otherwise.
  L.append(x)
    Append x to the end of the list L.
  L.index(value) -> integer
    Returns the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.remove(value)
    Removes the first occurrence of value from L.
  L.reverse()
    Reverse *IN PLACE*
  L.sort()
    Sorts the list in ascending order.
random:
  randint(a, b)
    Return random integer in range [a, b], including both end points.
str:
  x in s --> boolean
    Return True if x is in s and False otherwise.
  str(x) -> string
    Convert an object into its string representation, if possible.
  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.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.index(sub) -> integer
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() -> boolean
    Return True if all characters in S are digits and False otherwise.
  S.lower() -> string
    Return a copy of the string S converted to lowercase.
  S.lstrip([chars]) -> string
    Return a copy of the string S with leading whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.replace(old, new) -> string
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.rstrip([chars]) -> string
    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]) -> list of strings
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() -> string
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() -> string
    Return a copy of the string S converted to uppercase.
```