USN | | | | | | | | | | |

Internal Assessment Test II – April 2019

Sub:	Python Application Programming					Sub Code:	15CS664	Branch:	ISE/ECE/TCE
Date:	20/04/2019	Duration:	90 mins	Max Marks:	50	Sem/Sec :	6th Sem Open Elective		OBE

<u>Answer any **FIVE** FULL Questions</u> — MARKS | CO | RBT

1 (a) Explain string slices with the help of code snippets [04] CO2 L1

Scheme:

Definition – 1M
Different examples – 3M

Solution:

- **A segment of a string is called a slice.**
- **For Example:**

```
>>> s = 'Leela Palace'
>>> print(s[0:4])
Leel
>>> print(s[0:5])
Leela
>>> print(s[0:6])
Leela
>>> print(s[0:7])
Leela P
>>> print(s[2:9])
ela Pal
>>> print(s[-1:-7])      → This won't work.
                         → It results in an empty string
>>> print(s[-1])
e
>>> print(s[-4])
l
```

The operator returns the part of the string from the "n-eth" character to the "m-eth" character, including the first but excluding the last.

- **If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string.**

```
>>> veg = 'babycorn'
>>> print(veg[:4])
baby
```

```
>>> print(veg[4:])
corn
```

➢ If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks.

```
>>> fruit = 'apple'
>>> print(fruit[2:2])
                                → Empty String
>>> print(fruit[-1:0])
                                → Empty String
>>> print(fruit[-3:-1])
pl
>>> print(fruit[-5:-1])
appl
>>> print(fruit[-5:0])
                                → Empty String
>>>
>>> print(fruit[:])             → Exercise 2
Apple
```

Note: An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

(b)	Briefly explain any 6 built in functions in strings with examples.	[06]	CO2	L1

Scheme:

Any 6 built in function with syntax and example – 1*6M

Solution:

1. **upper()**
   - The upper() method returns the uppercased string from the given string. It converts all lowercase characters to uppercase.
   - If no lowercase characters exist, it returns the original string.
   - For example:

     ```
     >>> str = 'python application programming'
     >>> print(str.upper())
     PYTHON APPLICATION PROGRAMMING
     ```

2. **find()**
   - Syntax:

     ```
     str.find(sub[, start[, end]] )
     ```

sub - It's the substring to be searched in the *str* string.
start (optional) - starting index, by default its 0.
end (optional) – ending index, by default its equal to the length of the string
Note: [] means optional.

- **The find() method returns an integer value:**
  - ✓ **If substring exists inside the string, it returns the lowest index where substring is found.**
  - ✓ **If substring doesn't exist inside the string, it returns -1.**
- **For example:**

I.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('old'))
14
>>> quote = 'it is not too Old and it is not too late'
>>> print(quote.find('old'))
-1
```

II.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('o',10))
11
>>> print(quote.find('o',15))
29
```

III.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('too',9,30))
10
>>> print(quote.find('too',10,30))
10
>>> print(quote.find('too',20,30))
-1
>>> print(quote.find('too',30,20))
-1
>>> print(quote.find('too',20,40))
32
```

IV.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('and',-40,-10))
18
>>> print(quote.find('and',-10,-40))
```

-1

3. strip()

- **Syntax:**

  `str.strip( [chars] )`

  chars (optional) - a string specifying the set of characters to be removed. If the *chars* argument is not provided, all leading and trailing whitespaces are removed from the string.

- The strip() returns a copy of the string with both leading and trailing characters stripped.
  - ✓ When the combination of characters in the *chars* argument mismatches the character of the string in the left, it stops removing the leading characters.
  - ✓ When the combination of characters in the *chars* argument mismatches the character of the string in the right, it stops removing the trailing characters.
- **For example:**

I.
```
>>> string = '   welcome to python programming   '
>>> print(string.strip())
welcome to python programming
```

II.
```
>>> string = 'welcome to python programming'
>>> print(string.strip())
welcome to python programming
```

III.
```
>>> string = '***welcome ** to * python programming***'
>>> print(string.strip('*'))
welcome ** to * python programming
```

IV.
```
>>> string = 'welcome to python programming'
>>> print(string.strip('welcome'))
 to python programming
```

V.
```
>>> string = 'welcome to python programming'
>>> print(string.strip('welcome to '))
python programming
```

VI.
```
>>> string = 'welcome to python programming'
>>> print(string.strip('gam'))
welcome to python programmin
```

4. startswith()

- **Syntax:**

  `str.startswith(prefix[, start[, end]])`

- The startswith() method takes maximum of three parameters:
  - ✓ **prefix** - String or tuple of strings to be checked.

- ✓ **start** (optional) - Beginning position where **prefix** is to be checked within the string.
- ✓ **end** (optional) - Ending position where **prefix** is to be checked within the string.

- The startswith() method returns a boolean:
  - ✓ It returns *True* if the string starts with the specified prefix.
  - ✓ It returns *False* if the string doesn't start with the specified prefix.

- **For example:**

I.
```
>>> text = 'Python is easy to learn'
>>> res = text.startswith('python')
>>> print(res)
False
```

II.
```
>>> text = 'Python is easy to learn'
>>> res = text.startswith('Python is easy')
>>> print(res)
True
```

III.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming',7)
>>> print(res)
True
```

IV.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming',8)
>>> print(res)
False
```

V.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming is',7,18)
>>> print(res)
False
```

VI.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('program',7,18)
>>> print(res)
True
```

VII.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('easy to',7,18)
>>> print(res)
False
```

5. lower()

- **The lower() method returns the lowercased string from the given string. It converts all uppercase characters to lowercase.**
- **If no uppercase characters exist, it returns the original string.**
- **For example:**

  I. **>>> string = 'THIS SHOULD BE IN LOWERCASE'**
  >>> print(string.lower())
  this should be in lowercase

  II. **>>> string = 'th!s shouLd3 Be iN lOwer#case'**
  >>> print(string.lower())
  th!s should3 be in lower#case

Syntax	Meaning	Return Value
`str.count(substring[, start[,end]])`	count() method only requires a single parameter for execution. However, it also has two optional parameters: • **substring** - string whose count is to be found. • **start (Optional)** - starting index within the string where search starts. • **end (Optional)** - ending index within the string where search ends.	count() method returns the number occurrences of the substring in the given strir
`str.index(sub[, start[, end]])`	The index() method takes three parameters: • **sub** - substring to be searched in the string *str*. • **start** and **end**(optiona	• If substring exists inside the string, returns the lowest index in the stri where substring is found. • If substring doesn't exist inside string, it raises a **ValueError** excepti The index() method is similar to fin method for strings.

	l) - substring is searched within **str[start:end]**	The only difference is that *find()* meth returns -1 if the substring is not found, where *index()* throws an exception.
str.isalnum()	-	The isalnum() returns: • **True** if all characters in the string alphanumeric • **False** if at least one character is alphanumeric
str.isalpha()	-	The isalpha() returns: • **True** if all characters in the string alphabets (can be both lowercase a uppercase). • **False** if at least one character is alphabet.
str.isdecimal()	-	The isdecimal() returns: • **True** if all characters in the string decimal characters. • **False** if at least one character is decimal character.
str.isdigit()	-	The isdigit() returns: • **True** if all characters in the string are digits. • **False** if at least one character is not a digit.
str.islower()	-	The islower() method returns: • *True* if all alphabets that exist in string are lowercase alphabets. • *False* if the string contains at least c uppercase alphabet.
str.swapcase()	-	The swapcase() method returns the string wh all uppercase characters are converted lowercase, and lowercase characters converted to uppercase.
str.replace(old, new [, count])	The replace() method can take maximum of 3 parameters: • **old** - old substring you want to replace • **new** - new substring which would replace the old substring • **count** (optional) -	The replace() method returns a copy of string where *old* substring is replaced with *new* substring. The original string is unchang If the *old* substring is not found, it retu the copy of the original string.

	the number of times you want to replace the *old* substring with the *new* substring If *count* is not specified, replace() method replaces all occurrences of the *old* substring with the *new* substring.	

`str.join(iterable)`	The join() method takes an iterable - objects capable of returning its members one at a time
	Some of the example of iterables are:
	• **Native datatypes** - List, Tuple, String, Dictionary and Set
	• File objects and objects you define with an __iter__() or __getitem()__ method
		The join() method returns a stri concatenated with the elements of an iterable
		If the iterable contains any non-stri values, it raises a **TypeError** exception.
		Examples:
		1. >>> numList = ['1','2','3','4']
		>>> separator = ','
		>>> print(separator.join(numList))
		1,2,3,4
		2. >>> numTuple = ('1', '2', '3', '4')
		>>> separator = ','
		>>> print(separator.join(numTuple))
		1,2,3,4
		3. >>> str1 = 'anushka'
		>>> str2 = 'virat'
		>>> print(str1.join(str2))
		vanushkaianushkaranushkaaanushka
		>>> print(str2.join(str1))
		aviratnviratuviratsvirathviratkvirata

2 (a) What is file handle ? Explain the importance of file handle and its methods in handling files. [04] CO2 L2

Scheme:

Definition of file handle – 1M
Importance of file handle – 1M
Any two file methods – 2M

Solution:

- Create a file sample.txt (File→ New File→ sample.txt)

- We open a file sample.txt, which should be stored in the same folder that you are in when you start Python. By default it will open in read mode.

```
>>> fhand = open('sample.txt')
>>> print(fhand)
<_io.TextIOWrapper name='sample.txt' mode='r' encoding='cp1252'>
```

Note: Refer https://docs.python.org/3/library/io.html for more details regarding io.TextIOWrapper

- If the open is successful, the operating system returns us a *file handle*. The file
handle is not the actual data contained in the file, but instead it is a “handle” that
we can use to read the data. You are given a handle if the requested file exists and
you have the proper permissions to read the file.

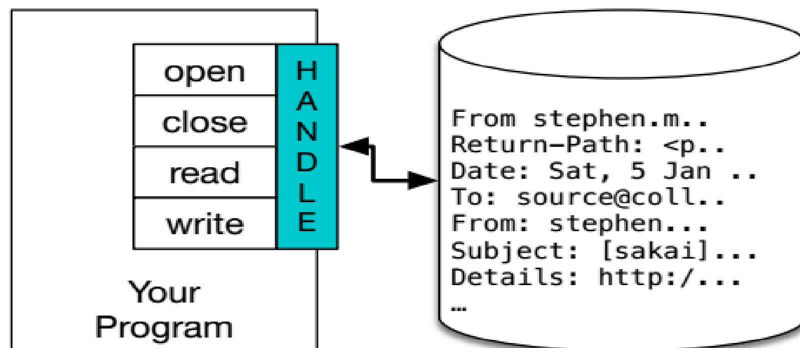

Fig: A file handle

- If the file does not exist, open will fail with a traceback and you will not get a handle to access the contents of the file:

```
>>> fhand = open('test.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

- If you know the file is relatively small compared to the size of your main memory, you can read the whole file into one string using the read method on the file handle.

```
>>> fhand = open('poem.txt')
>>> string = fhand.read()
>>> print(len(string))
611
>>> print(string[:30])
```

When to the session of sweet s

- To write a file, you have to open it with mode “w” as a second parameter:

```
>>> fout = open('wfile.txt','w')
>>> print(fout)
<_io.TextIOWrapper name='wfile.txt' mode='w' encoding='cp1252'>
```

- **If the file already exists, opening it in write mode clears out the old data and starts fresh. If the file doesn’t exist, a new one is created.**
- The write method of the file handle object puts data into the file, returning the number of characters written. The default write mode is text for writing (and reading) strings.

```
>>> mess = 'Hi. Good Morning All!!!'
>>> fout.write(mess)
23
```

- After opening a file one should always close the opened file. We use method *close()* for this.
- **For example:**

```
>>> fhand = open('sample.txt')
>>> fhand.readlines()
['Welcome to Python Application Programming.\n', 'This is Akhilaa your course instructor.\n']
>>> fhand.close()
```

- Always make sure you *explicitly* close each opened file, once its job is done and you have no reason to keep it open. Because - There is an upper limit to the number of files a program can open. If you exceed that limit, there is no reliable way of recovery, so the program could crash. – Each open file consumes some main-memory for the data-structures associated with it, like file descriptor/handle or file locks etc. So you could essentially end-up wasting lots of memory if you have more files open that are not useful or usable. - Open files always stand a chance of corruption and data loss.

(b) Write a program in python to print all the lines which has email domain address (ex: @utc.ac.za/gmail.com) in **Sample-file.txt** [06] CO2 L2

Sample-file.txt

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za …

Scheme:

Input – 1M
Scanning line starting and splitting into words – 2M
Extracting mail address – 2M
Output – 1M

Solution:

```
fhand = open('sample-file.txt')

for line in fhand:
   line = line.rstrip()
   if not line.startswith('From '):
      continue
   words = line.split()
   eadd = words[2]
   atpos = data.find('@')
   spos = data.find(' ', atpos)
   host = data[atpos+1:spos]
   print(host)
```

3 (a) Bring out the difference between the following with examples [10] CO3 L2

a. lists, dictionaries and tuples.
b. append() and extend().
c. pop() and remove()
d. find() and startswith()

Scheme:

Syntax and example for each function : 2.5 * 4

Solution:

a.

List	Tuple
The literal syntax of lists is shown by square brackets []	The literal syntax of tuples is shown by parentheses ()
Lists are mutable	Tuples are immutable
Lists have order	Tuples have structures
Lists are for variable length	Tuples are for fixed length
Lists can be indexed, sliced and compared	Tuples can be indexed, sliced and compared
List are usually homogenous	Tuple are usually heterogeneous
Iterating through a list is slower compared to tuple	Iterating through a tuple is faster
Lists cannot be used as key in dictionary	Tuples can be used as a key in dictionary

- **A *dictionary* is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.**
- **We can think of a dictionary as a mapping between a set of indices (which are called *keys*) and a set of values. Each key maps to a value. The association of a key and a value is called a *key-value pair* or sometimes an *item*.**

b.

- ✓ **The pop() method removes and returns the element at the given index (passed as an argument) from the list.**
- ✓ The syntax is:

```
list_name.pop(index)
```

- ✓ The pop() method takes a single argument (index) and removes the element present at that index from the list.
- ✓ If the index passed to the pop() method is not in the range, it throws **IndexError: pop index out of range** exception.
- ✓ The parameter passed to the pop() method is optional. If no parameter is passed, the default index **-1** is passed as an argument which returns the last element.
- ✓ The pop() method returns the element present at the given index.
- ✓ Also, the pop() method removes the element at the given index and updates the list.
- ✓ **For example:**

I.

```
>>> prolan = ['Python', 'Java', 'C', 'C++','PHP']
>>> prolan.pop()
'PHP'
>>> print(prolan)
['Python', 'Java', 'C', 'C++']
>>> res = prolan.pop(1)
>>> print(res)
Java
>>> print(prolan)
['Python', 'C', 'C++']
>>> res = prolan.pop(6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop index out of range
>>> res = prolan.pop(-2)
>>> print(prolan)
['Python', 'C++']
```

II.

```
>>> mix = ['awesome', 19, 27.5, [14, 32]]
>>> mix.pop(3)
[14, 32]
```

- ✓ The remove() method searches for the given element in the list and removes the first matching element.
- ✓ The syntax is:

```
list_name.remove(element)
```

- ✓ The remove() method takes a single element as an argument and removes it from the list.
- ✓ If the **element**(argument) passed to the remove() method doesn't exist, **valueError** exception is thrown.
- ✓ The remove() method only removes the given element from the list. It doesn't return any value.
- ✓ **For example:**

I.
```
>>> mix = ['awesome', 19, 27.5, [14, 32]]
>>> mix.remove(19)
>>> print(mix)
['awesome', 27.5, [14, 32]]
>>> mix.remove(141)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

II.
```
>>> lang = ['Kannada', 'Telugu', 'Tamil', 'Tamil', 'Hindi', 'English']
>>> lang.remove('Tamil')
>>> print(lang)
['Kannada', 'Telugu', 'Tamil', 'Hindi', 'English']
```

c.

- ✓ **The append() method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list.**
- ✓ The syntax is:

```
list_name.append(item)
```

- ✓ The append() method takes a single *item* and adds it to the end of the list. The *item* can be numbers, strings, another list, dictionary etc.
- ✓ **For example:**

I.
```
>>> n = [1, 2, 3, 4]
>>> n.append('end')
>>> print(n)
[1, 2, 3, 4, 'end']
>>> n = [1, 2, 3, 4]
```

II.
```
>>> n.append([5, 6])
>>> print(n)
[1, 2, 3, 4, [5, 6]]
```

- ✓ **The extend() extends the list by adding all items of a list (passed as an argument) to the end.**
- ✓ The extend() method takes a single argument (a list) and adds it to the end.
- ✓ The syntax is:

```
list1_name.extend(list2_name)
```

- ✓ **For example:**

```
>>> branch = ['ise', 'cse', 'tce','ece']
>>> branch1 = ['mech', 'civil']
>>> branch.extend(branch1)
>>> print(branch)
['ise', 'cse', 'tce', 'ece', 'mech', 'civil']
```

d.

- **Syntax:**

```
str.find(sub[, start[, end]] )
```

sub - It's the substring to be searched in the *str* string.
start (optional) - starting index, by default its 0.
end (optional) – ending index, by default its equal to the length of the string
Note: [] means optional.

- **The find() method returns an integer value:**
  - ✓ **If substring exists inside the string, it returns the lowest index where substring is found.**
  - ✓ **If substring doesn't exist inside the string, it returns -1.**
- **For example:**

V.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('old'))
14
>>> quote = 'it is not too Old and it is not too late'
>>> print(quote.find('old'))
-1
```

VI.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('o',10))
11
>>> print(quote.find('o',15))
29
```

VII.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('too',9,30))
10
>>> print(quote.find('too',10,30))
10
>>> print(quote.find('too',20,30))
-1
>>> print(quote.find('too',30,20))
-1
>>> print(quote.find('too',20,40))
32
```

VIII. **>>> quote = 'it is not too old and it is not too late'**

>>> print(quote.find('and',-40,-10))

18

>>> print(quote.find('and',-10,-40))

-1

- **Syntax:**

```
str.startswith(prefix[, start[, end]])
```

- The startswith() method takes maximum of three parameters:
  - ✓ **prefix** - String or tuple of strings to be checked.
  - ✓ **start** (optional) - Beginning position where **prefix** is to be checked within the string.
  - ✓ **end** (optional) - Ending position where **prefix** is to be checked within the string.
- The startswith() method returns a boolean:
  - ✓ It returns *True* if the string starts with the specified prefix.
  - ✓ It returns *False* if the string doesn't start with the specified prefix.

- **For example:**

VIII. **>>> text = 'Python is easy to learn'**

>>> res = text.startswith('python')

>>> print(res)

False

IX. **>>> text = 'Python is easy to learn'**

>>> res = text.startswith('Python is easy')

>>> print(res)

True

X. **>>> text = 'Python programming is easy'**

>>> res = text.startswith('programming',7)

>>> print(res)

True

XI. **>>> text = 'Python programming is easy'**

>>> res = text.startswith('programming',8)

>>> print(res)

False

XII. **>>> text = 'Python programming is easy'**

>>> res = text.startswith('programming is',7,18)

>>> print(res)

False

XIII. **>>> text = 'Python programming is easy'**

>>> res = text.startswith('program',7,18)

>>> print(res)

True

```
XIV. >>> text = 'Python programming is easy'
     >>> res = text.startswith('easy to',7,18)
     >>> print(res)
     False
```

4 (a)	Write a program in python to prompt the user to enter the number from console multiple times, until user enters 'done'. Once 'done' is entered, find and display the average of the these numbers which is stored in the list. (Note: Use built in functions of lists to find the average).	[7]	CO3	L3

Scheme:

Input – 1M
Logic correctness – 4M
Output – 2M

Solution:

```
num = []                # or num = list()
while(True):
   n = input('Enter a number: ')
   if n == 'done':
      break
   val = float(n)
   num.append(val)

avg = sum(num)/len(num)
print('The average is:', avg)
```

(b)	Consider a string "**pining*for*the fjords**", explain how to split the string based on * and after splitting join them back with delimiter --- **Final Output : pining---for---the fjords.**	[3]	CO3	L2

Scheme:

Each step – 1M

Solution:

```
str = "pining*for*the fjords"
t = str.split('*')
delimeter = '---'
delimeter.join(t)
```

5 (a)	What are dictionaries in python? Write a program to display the total count of each character in string '**brontosaurus**' using dictionaries. **Output : {'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}**	[10]	CO3	L2

Scheme:

Definition – 2M
Input – 1M
Logic – 5M
Output – 2M

Solution:

- **A *dictionary* is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.**
- **We can think of a dictionary as a mapping between a set of indices (which are called *keys*) and a set of values. Each key maps to a value. The association of a key and a value is called a *key-value pair* or sometimes an *item*.**

```
word = input('Enter the string: ')
d = dict()
for c in word:
   if c not in d:
      d[c] = 1
   else:
      d[c] = d[c] + 1
print(d)
```

		Marks	CO	RBT
6 (a)	Write a program in python to get the following output.	[10]	CO3	L3

Enter the file name : **Romeo.txt**
The count of each word are as follows
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1, 'is': 3, 'through': 1, 'pale': 1, 'yonder': 1, 'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1, 'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1, 'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}

Note: Romeo.txt contains the following paragraph
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief

Scheme:

Input: 2M
For loop for handling lines in file – 3M
For loop for finding the frequency – 4M
Output – 1M

Solution:

```
fname = input('Enter the file name: ')
try:
```

```
        fhand = open(fname)
    except:
        print('File cannot be opened:', fname)
        exit()

    counts = dict()
    for line in fhand:
        words = line.split()
        for word in words:
            if word not in counts:
                counts[word] = 1
            else:
                counts[word] += 1

    print(counts)
```

7 (a)	Briefly explain DSU pattern. Using DSU pattern, write a program to sort the words in a string 'but soft what light in yonder window breaks' from longest to shortest.	[10]	CO3	L2

Scheme:

DSU Explanation - 3M
Input – 1M
For loop to append words – 2M
Reverse the list – 1M
Put the result in a list – 2M
Output – 1M

Solution:

- The sort function works the same way. It sorts primarily by first element, but in the case of a tie, it sorts by second element, and so on. This feature lends itself to a pattern called **DSU** for
  - **Decorate** a sequence by building a list of tuples with one or more sort keys preceding the elements from the sequence,
  - **Sort** the list of tuples using the Python built-in sort, and
  - **Undecorate** by extracting the sorted elements of the sequence.

```
txt = 'but soft what light in yonder window breaks'

words = txt.split()
t = list()

for word in words:
    t.append((len(word), word))
```

```
print('\nThe list is:\n',t)
t.sort(reverse=True)
print('\nThe list after sorting is:\n',t)

res = list()

for length, word in t:
   res.append(word)

print('\nThe sorted list is:\n',res)
```