MODULE 3

1 LISTS

Like a string, **a *list* is a sequence of values**. In a string, the values are characters; in a list, they **can be any type. The values in list are called *elements* or sometimes *items*.**

1.1 CREATING A LIST

- A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

  For example,

  I. **num = [10, 20, 30, 40, 50]→ List of five integers**
  II. **string = ['Virat Kohli', 'MS Dhoni', 'Hardik Pandya', 'Sachin Tendulkar']→ List of four strings**

- The elements of a list don't have to be the same type.

  For example,

  mix = ['awesome', 19, 27.5, [14, 32]]

  The above list contains a string, an integer, a float and a nested list. A list within a list is called a nested list

- **A list that contains no elements is called an empty list; we can create one with empty brackets, [].**

  For example,

  list = []

- We can print the list as follows:

  >>> print(list,mix,string,num)

  [] ['awesome', 19, 27.5, [14, 32]] ['Virat Kohli', 'MS Dhoni', 'Hardik Pandya', 'Sachin Tendulkar'] [10, 20, 30, 40, 50]

1.2 ACCESSING ELEMENTS FROM A LIST

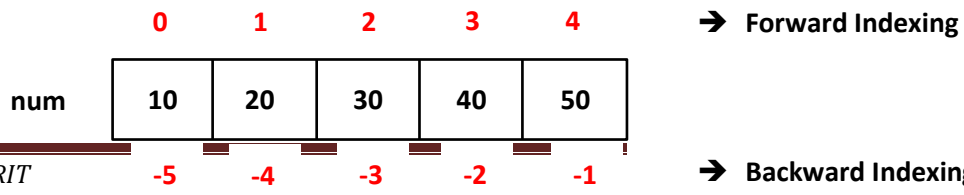
- The list can be accessed from both the directions in forward and backward.

  Forward indexing starts with 0,1,2,3,....

  Backward indexing starts with -1,-2,-3,-4,....

- We can **access the characters one at a time with the bracket operator ([])**.
- **For example: Consider a list containing 5 numbers.**

num = [10, 20, 30, 40, 50]



- In the above figure num[0] gives the first number, num[1] gives second number and so on. Similarly num[-1] gives the last number, num[-2] gives the last but one number and so on.
- **For example:**

1. **List Index/Forward Indexing**
   - **We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.**
   - **Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.**
   - **For example:**

a.
```
>>> list = ['P', 'Y', 'T', 'H', 'O', 'N']
>>> print(list[1])
Y
>>> print(list[3])
H
>>> print(list[4])
O
>>> print(list[6])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> print(list[2.2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not float
```

b.
```
>>> n_list = ['Beautiful',[2,0,1,8]]
>>> print(n_list[0][3])
u
>>> print(n_list[1][3])
8
>>> print(n_list[1][0])
2
>>> print(n_list[0][5])
i
```

2. **Negative Indexing/Backward Indexing**
   - **Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.**
   - **For example:**

a.
```
>>> list = ['P', 'Y', 'T', 'H', 'O', 'N']
>>> print(list[-1])
N
>>> print(list[-6])
```

```
P
>>> print(list[-3])
H
```

b.
```
>>> n_list = ['Beautiful',[2,0,1,8]]
>>> print(n_list[-1][3])
8
>>> print(n_list[-1][-1])
8
>>> print(n_list[-1][-4])
2
>>> print(n_list[-2][-1])
l
>>> print(n_list[-2][-5])
t
```

1.3 LISTS ARE MUTABLE

- Unlike strings, **lists are mutable because you can change the order of items in a list or reassign an item in a list.**

  For example:

```
>>> num = [10, 20, 30]
>>> num[1] = 40
>>> print(num)
[10, 40, 30]
```

- We can think of a **list as a relationship between indices and elements. This relationship is called a *mapping*; each index "maps to" one of the elements.**
- List indices work the same way as string indices:
  - Any integer expression can be used as an index.
  - If you try to read or write an element that does not exist, you get an IndexError.
  - If an index has a negative value, it counts backward from the end of the list.
- The in operator also works on lists.

  For example:

```
>>> num = [10, 20, 30]
>>> 30 in num
True
>>> 50 in num
False
```

1.4 TRAVERSING A LIST

- The most common way to traverse the elements of a list is with a for loop.

  For example:

```
>>> player = ['Virat Kohli', 'MS Dhoni', 'Hardik Pandya', 'Sachin Tendulkar']
```

```
>>> for name in player:
...     print(name)
...
Virat Kohli
MS Dhoni
Hardik Pandya
Sachin Tendulkar
```

- This works well if you only need to read the elements of the list. But if you want to write or update the elements, you need the indices. A common way to do that is to combine the functions range and len.

For example:

```
>>> num = [10, 20, 30]
>>> for i in range(len(num)):
...     num[i] = num[i] * 2
...
>>> print(num)
[20, 40, 60]
```

- This loop traverses the list and updates each element.
- len returns the number of elements in the list.
- range returns a list of indices from 0 to *n* - 1, where *n* is the length of the list.
- Each time through the loop, i gets the index of the next element.
- The assignment statement in the body uses i to read the old value of the element and to assign the new value.

- Although a list can contain another list, the nested list still counts as a single element.

For example:

```
>>> mix = ['awesome', 19, 27.5, [14, 32]]
>>> len(mix)
4
```

1.5 LIST OPERATIONS

- **'+' and '*' operators can be used on lists.**
- **The '+' operator concatenates lists.**

For example:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
```

- **The '*' operator repeats a list given number of times.**

For example:

1. **>>> a = [1, 2, 3] *3**
   >>> print(a)
   [1, 2, 3, 1, 2, 3, 1, 2, 3]
2. **>>> a = [1, 2, 3] * -1**
   >>> print(a)
   []
3. **>>> a = [1, 2, 3] * 0**
   >>> print(a)
   []

1.6 LIST SLICES

- The slice operator also works on lists.
- If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. So if you omit both, the slice is a copy of the whole list.

For example:

```
>>> s = ['a', 'b', 'c', 'd', 'e', 'f']
>>> s[2:4]
['c', 'd']
>>> s[:4]
['a', 'b', 'c', 'd']
>>> s[2:]
['c', 'd', 'e', 'f']
>>> s[:]
['a', 'b', 'c', 'd', 'e', 'f']
>>> s[-2:-4]
[]
>>> s[-4:-2]
['c', 'd']
>>> s[-4:]
['c', 'd', 'e', 'f']
>>> s[:-1]
['a', 'b', 'c', 'd', 'e']
>>> s[-1:]
['f']
>>> s[-5:]
['b', 'c', 'd', 'e', 'f']
>>> s[:-4]
['a', 'b']
```

- A slice operator on the left side of an assignment can update multiple elements.

For example:

```
>>> s = ['a', 'b', 'c', 'd', 'e', 'f']
>>> s[2:4] = ['p', 'q']
>>> print(s)
['a', 'b', 'p', 'q', 'e', 'f']
```

1.7 LIST METHODS

- Python provides methods that operate on lists.

1. **append()**
   - ✓ **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]]
     ```

2. **extend()**
   - ✓ **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']
     ```

3. **sort()**
   - ✓ **The sort() method sorts the elements of a given list.**
   - ✓ The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.
   - ✓ The syntax is:

     `list_name.sort()`

- ✓ By default, sort() doesn't require any extra parameters. However an optional parameter: **reverse** - **If true, the sorted list is reversed (or sorted in Descending order)**
- ✓ **For example:**

I.
```
>>> vowels = ['u', 'i', 'a', 'e', 'o']
>>> vowels.sort()
>>> print(vowels)
['a', 'e', 'i', 'o', 'u']
```

II.
```
>>> vowels = ['u', 'i', 'a', 'e', 'o']
>>> vowels.sort(reverse=True)
>>> print(vowels)
['u', 'o', 'i', 'e', 'a']
```

4. **insert()**
   - ✓ **The insert() method inserts the element to the list at the given index.**
   - ✓ The syntax is:

```
list_name.insert(index, element)
```

   - ✓ The insert() function takes two parameters:
     - **index** - position where element needs to be inserted
     - **element** - this is the element to be inserted in the list
   - ✓ **For example:**

I.
```
>>> vowels.insert(2, 'i')
>>> print(vowels)
['a', 'e', 'i', 'o', 'u']
```

II.
```
>>> num  = [34, 56, 67, 89]
>>> num.insert(6, 23)
>>> print(num)
[34, 56, 67, 89, 23]
```

III.
```
>>> num  = [34, 56, 67, 89]
>>> num.insert(0, 23)
>>> print(num)
[23, 34, 56, 67, 89]
>>> num.insert(-1, 23)
>>> print(num)
[23, 34, 56, 67, 23, 89]
```

5. **index()**
   - ✓ **The index() method searches an element in the list and returns its index.**
   - ✓ The syntax is:

```
list_name.index(element)
```

   - ✓ The index method takes a single argument: **element** - element that is to be searched.
   - ✓ If the same element is present more than once, index() method returns its smallest/first position.

- ✓ The index() method returns the index of the element in the list.
- ✓ If not found, it raises a ValueError exception indicating the element is not in the list.
- ✓ **For example:**

```
>>> list = [85, -5, 546, 43, -532, 32]
>>> i = list.index(43)
>>> print(i)
3
>>> i = list.index(9)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 9 is not in list
```

6. **reverse()**
   - ✓ **The reverse() method reverses the elements of a given list.**
   - ✓ The syntax is:

```
list_name.reverse()
```

   - ✓ **For example:**

```
>>> os = ['iOS', 'Android', 'Windows', 'Linux']
>>> os.reverse()
>>> print(os)
['Linux', 'Windows', 'Android', 'iOS']
>>> os[::-1]
['iOS', 'Android', 'Windows', 'Linux']
```

   - ✓ Accessing individual elements in reversed order:

```
>>> os = ['iOS', 'Android', 'Windows', 'Linux']
>>> for i in reversed(os):
...     print(i)
...
Linux
Windows
Android
iOS
```

7. **count()**
   - ✓ The count() method returns the number of occurrences of an element in a list.
   - ✓ The syntax is:

```
list_name.count(element)
```

   - ✓ The count() method takes a single argument: **element** - element whose count is to be found.
   - ✓ The count() method returns the number of occurrences of an element in a list.
   - ✓ **For example:**

```
>>> n = [1, 2, 2, 3, 4, 5, 5]
>>> n.count(2)
```

```
2
>>> n.count(4)
1
>>> n.count(8)
0
```

8. **clear()**
   - ✓ **The clear() method removes all items from the list.**
   - ✓ The syntax is:

```
list_name.clear()
```

   - ✓ **For example:**

```
>>> os = ['iOS', 'Android', 'Windows', 'Linux']
>>> os.clear()
>>> print(os)
[]
```

9. **copy()**
   - ✓ **The copy() method returns a shallow copy of the list.**
   - ✓ A list can be copied with = operator.

   For example:

```
>>> n = [1, 2, 3]
>>> new = n
>>> print(new)
[1, 2, 3]
>>> print(n)
[1, 2, 3]
```

   - ✓ The problem with copying the list in this way is that if you modify the *new*, the *n* is also modified.

   For example:

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

   - ✓ **If you need the original list unchanged when the new list is modified, you can use copy() method. This is called shallow copy.**
   - ✓ The syntax is:

```
new_list = list.copy()
```

   - ✓ The copy() function returns a list. It doesn't modify the original list.
   - ✓ **For example:**

```
>>> mix = ['awesome', 19, 27.5]
>>> new = mix.copy()
```

```
>>> new.append([34, -143])
>>> print(mix)
['awesome', 19, 27.5]
>>> print(new)
['awesome', 19, 27.5, [34, -143]]
```

1.8 DELETING ELEMENTS

- There are several ways to delete elements from a list.

1. **pop()**
   - ✓ **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]
```

2. **del**
   - ✓ If you don't need the removed value, you can use the del operator.
   - ✓ The syntax is:

```
del list_name[index_val]
```

   - ✓ **For example:**

```
>>> n = [10, 52, 37]
>>> del n[1]
>>> print(n)
[10, 37]
```

3. **remove()**
   - ✓ 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']
```

1.9 LISTS AND FUNCTIONS

- There are a number of built-in functions that can be used on lists that allow you to quickly look through a list without writing your own loops.

1. **len()**
   - ✓ **The len() function returns the number of items (length) of an object.**
   - ✓ The syntax is:

```
len(s)
```

- ✓ **s** - a sequence (string, bytes, tuple, list, or range) or a collection (dictionary, set or frozen set)
- ✓ The len() function returns the number of items of an object.
- ✓ **For example:**

I.
```
>>> num = []
>>> print(len(num))
0
>>> num = [1, 2, 3, 4, 5]
>>> print(len(num))
5
```

II.
```
>>> mix = ['awesome', 19, 27.5, [14, 32]]
>>> print(len(mix))
4
```

2. min()

- ✓ The max() method returns the smallest element of two or more parameters.
- ✓ The syntax is:

```
min(list)
```

- ✓ **For example:**

I.
```
>>> num = [1, 2, 3, 4, 5]
>>> print(min(num))
1
```

II.
```
>>> vow = ['e', 'a', 'u', 'i', 'o']
>>> print(min(vow))
a
```

III.
```
>>> l = ['eert', 'aert', 'urty', 'iery', 'odh']
>>> print(min(l))
aert
```

IV.
```
>>> mix = ['awesome', 19, 27.5, [14, 32]]
>>> print(min(mix))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

3. max()

- ✓ The max() method returns the largest element of two or more parameters.
- ✓ The syntax is:

```
min(list)
```

- ✓ **For example:**

I.
```
>>> num = [1, 2, 3, 4, 5]
>>> print(max(num))
5
```

II.
```
>>> vow = ['e', 'a', 'u', 'i', 'o']
```

```
>>> print(max(vow))
u
```

III.
```
>>> l = ['eert', 'aert', 'urty', 'iery', 'odh']
>>> print(max(l))
urty
```

4. **sum()**
   - ✓ **The sum() function adds the items of an iterable and returns the sum.**
   - ✓ The syntax is:

```
sum(iterable, start)
```

     - **iterable** - iterable (list, tuple, dict etc) whose item's sum is to be found. Normally, items of the iterable should be numbers.
     - **start** (optional) - this value is added to the sum of items of the iterable. The default value of *start* is 0 (if omitted)
   - ✓ The sum() function adds *start* and items of the given *iterable* from left to right.
   - ✓ The sum() function returns the sum of *start* and items of the given *iterable*.
   - ✓ **Note: The sum() works only when the list elements are numbers.**
   - ✓ **For example:**

```
>>> num = [2.5, 3, 4, -5]
>>> print(sum(num))
4.5
>>> print(sum(num)/len(num))
1.125
>>> nsum = sum(num, 15)
>>> print(nsum)
19.5
```

➢ **Python program to compute average without a list.**

```
sum = 0
count = 0

while(True):
  n = input('Enter a number: ')
  if n == 'done':
    break
  val = float(n)
  sum = sum + val
  count = count + 1

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

Output:

```
Enter a number: 7
Enter a number: 52.0
Enter a number: -232
Enter a number: 13
Enter a number: 93
Enter a number: -9.4
Enter a number: 34.2
Enter a number: done
The average is: -6.028571428571429
```

- We could simply remember each number as the user entered it and use built-in functions to compute the sum and count at the end.

```
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)
```

Output:

```
Enter a number: 2
Enter a number: -134
Enter a number: -9.34
Enter a number: 834.2
Enter a number: 2
Enter a number: -143
Enter a number: 47
Enter a number: 0.13
Enter a number: done
The average is: 74.87375
```

1.10 LISTS AND STRINGS

- A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as a string. To convert from a string to a list of characters, you can use list:

For example:

```
>>> s = 'spam'
>>> t = list(s)
```

```
>>> print(t)
['s', 'p', 'a', 'm']
```

- Because list is the name of a built-in function, you should avoid using it as a variable name.
- The list function breaks a string into individual letters. If you want to break a string into words, you can use the split method. Once you have used split to break the string into a list of words, you can use the index operator (square bracket) to look at a particular word in the list.

For example:

```
>>> s = 'The Python Programming Language'
>>> t = s.split()
>>> print(t)
['The', 'Python', 'Programming', 'Language']
>>> print(t[1])
Python
```

- We can call split with an optional argument called a *delimiter* that specifies which characters to use as word boundaries. The following example uses a hyphen as a delimiter:

For example:

```
>>> s = 'u-get-back-whatever-you-give'
>>> delimeter = '-'
>>> s.split(delimeter)
['u', 'get', 'back', 'whatever', 'you', 'give']
```

- join is the inverse of split. It takes a list of strings and concatenates the elements. join is a string method, so you have to invoke it on the delimiter and pass the list as a parameter:

For example:

```
>>> t = ['Life', 'is', 'beautiful']
>>> delimeter = ' '
>>> delimeter.join(t)
'Life is beautiful'
```

- In this case the delimiter is a space character, so join puts a space between words. To concatenate strings without spaces, you can use the empty string, “”, as a delimiter.

For example:

```
>>> t = ['Life', 'is', 'beautiful']
>>> delimeter = ''
>>> delimeter.join(t)
'Lifeisbeautiful'
```

1.11 PARSING LINES

- When we are reading a file we want to do something to the lines other than just printing the whole line. Often we want to find the “interesting lines” and then *parse* the line to find some interesting

part of the line. What if we wanted to print out the day of the week from those lines that start with "From"?

From stephen.marquard@uct.ac.za *Sat* Jan 5 09:14:16 2008

- The split method is very effective when faced with this kind of problem. We can write a small program that looks for lines where the line starts with "From", split those lines, and then print out the third word in the line:

Note: Refer to Page 43 for file mbox.txt

```
fhand = open('mbox.txt')

for line in fhand:
   line = line.strip()
   if not line.startswith('From '):
      continue
   words = line.split()
   print(words[2])
```

Output:

```
Sat
Fri
Fri
Fri
Fri
```

1.12 OBJECTS AND VALUES

- Consider the following statements:

```
>>> a = 'banana'
>>> b = 'banana'
```

- We know that a and b both refer to a string, but we don't know whether they refer to the *same* string. There are two possible states:

Fig: Variables and Objects

In one case, a and b refer to two different objects that have the same value. In the second case, they refer to the same object.

- To check whether two variables refer to the same object, you can use the 'is' operator.

For example:

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
```

True

- In the above example, Python only created one string object, and both a and b refer to it. But when you create two lists, you get two objects:

  For example:

  ```
  >>> a = [1, 2, 3]
  >>> b = [1, 2, 3]
  >>> a is b
  False
  ```

- In this case we would say that the two lists are *equivalent*, because they have the same elements, but not *identical*, because they are not the same object. If two objects are identical, they are also equivalent, but if they are equivalent, they are not necessarily identical.
- Until now, we have been using "object" and "value" interchangeably, but it is more precise to say that an object has a value. If you execute a = [1, 2, 3], a refers to a list object whose value is a particular sequence of elements. If another list has the same elements, we would say it has the same value.

1.13 ALIASING

- If a refers to an object and you assign b = a, then both variables refer to the same object:

  For example:

  ```
  >>> a = [1, 2, 3]
  >>> b = a
  >>> b is a
  True
  ```

- The association of a variable with an object is called a *reference*. In this example, there are two references to the same object.
- An object with more than one reference has more than one name, so we say that the object is *aliased*.
- If the aliased object is mutable, changes made with one alias affect the other:

  ```
  >>> b[0] = 19
  >>> print(a)
  [19, 2, 3]
  ```

- Although this behaviour can be useful, it is error-prone. In general, it is safer to avoid aliasing when you are working with mutable objects.
- For immutable objects like strings, aliasing is not as much of a problem.

1.14 LIST ARGUMENTS

- **When you pass a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change.**

  For example:

  ```
  def remove_elem(a):
  ```

```
    del a[2]

a = [1, 3, 4, 6, 9]
remove_elem(a)
print(a)
```

Output:

[1, 3, 6, 9]

- It is important to distinguish between operations that modify lists and operations that create new lists. For example, the append method modifies a list, but the + operator creates a new list:

```
>>> t1 = [1, 2, 3]
>>> t2 = t1.append(4)
>>> print(t1)
[1, 2, 3, 4]
>>> print(t2)
None

>>> t3 = t1 + [3]
>>> print(t3)
[1, 2, 3, 4, 3]
>>> t2 is t3
False
```

- This difference is important when you write functions that are supposed to modify lists. For example, this function *does not* delete the head of a list:

```
def bad_delete_head(t):
  t = t[1:]      # WRONG!
```

The slice operator creates a new list and the assignment makes t refer to it, but none of that has any effect on the list that was passed as an argument.

- An alternative is to write a function that creates and returns a new list. For example, tail returns all but the first element of a list:

```
def tail(t):
  return t[1:]
```

This function leaves the original list unmodified. Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> print(rest)
['b', 'c']
```

1.15 EXERCISES

1. Write a python program to find the sum of elements in a list without using built in function sum().

2. Write a python program to find the largest and smallest element in a list without using built in functions max() and min().
3. Write a python program to find sum and average of elements in a list.
4. Write a python program to find the largest and smallest element in a list.
5. Write a python program to merge two lists and sort it.
6. Write a python program to convert a list of characters into string.
7. Write a python program to find frequency of elements in a given list.
8. Write a python program to put even and odd elements in a list into two different lists.
9. Write a python program to swap the first and last element of a list.
10. Write a python program to remove all the duplicate elements from a list.
11. Write a python program to read a list of words and return the length of the longest one.

1.16 GLOSSARY

aliasing	A circumstance where two or more variables refer to the same object.
delimiter	A character or string used to indicate where a string should be split.
element	One of the values in a list (or other sequence); also called items.
equivalent	Having the same value.
index	An integer value that indicates an element in a list.
identical	Being the same object (which implies equivalence).
list	A sequence of values.
list traversal	The sequential accessing of each element in a list.
nested list	A list that is an element of another list.
object	Something a variable can refer to. An object has a type and a value.
reference	The association between a variable and its value.

2 DICTIONARIES

- **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*.**
- **The function dict creates a new dictionary with no items. Because dict is the name of a built-in function, you should avoid using it as a variable name.**

For example:

```
>>> ele = dict()
>>> print(ele)
{}
```

OR

```
>>> ele = {}
>>> print(ele)
{}
```

2.1 CREATE A DICTIONARY

- Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.
- An item has a key and the corresponding value expressed as a pair, key: value.
- While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.
- The order of items in a dictionary is unpredictable.
- **For example:**

I.
```
>>> my_dict = {}          #Empty dictionary.
>>> print(ele)
{}
```

II.
```
>>> my_dict = dict({'name':'akhilaa', 'age':27})          #Using dict()
>>> print(my_dict)
{'name': 'akhilaa', 'age': 27}
```

III.
```
>>> my_dict = {1:'apple', 2:'bat'}          #Dictionary with integer keys.
>>> print(my_dict)
{1: 'apple', 2: 'bat'}
```

IV.
```
>>> my_dict = {'name':'akhilaa', 1:[10,20,30]}          #Dictionary with mixed keys.
>>> print(my_dict)
{'name': 'akhilaa', 1: [10, 20, 30]}
```

2.2 ACCESSING ELEMENTS FROM A DICTIONARY

- While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.
- The difference while using get() is that it returns None instead of KeyError, if the key is not found.
- **For example:**

I.
```
>>> my_dict = {'name':'akhilaa', 'age':27}
>>> print(my_dict['name'])
akhilaa
>>> print(my_dict['age'])
27
>>> print(my_dict['sal'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'sal'
>>> print(my_dict.get('name'))
akhilaa
>>> print(my_dict.get('age'))
27
>>> print(my_dict.get('sal'))
None
```

II.
```
>>> my_dict = {1:'Akhilaa', 'dob':[1,12,1990]}
>>> print(my_dict.get('dob'))
[1, 12, 1990]
>>> print(my_dict.get(1))
Akhilaa
>>> print(my_dict[‘dob’][1])
12
```

III.
```
>>> my_dict = {1:'Akhilaa', 2:[1,12,1990]}
>>> print(my_dict[2][1])
12
```

2.3 CHANGING OR ADDING ELEMENTS IN A DICTIONARY

- Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.
- If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.
- **For example:**

I.
```
>>> details = {'name':'arnitha', 'age':27}
>>> details['name'] = 'Akhilaa'
>>> print(details)
```

```
{'name': 'Akhilaa', 'age': 27}
>>> details['city'] = 'Bengaluru'
>>> print(details)
{'name': 'Akhilaa', 'age': 27, 'city': 'Bengaluru'}
```

II.
```
>>> my_dict = {1:'Akhilaa', 2:[1,12,1990]}
>>> my_dict[2][2] = 1988
>>> print(my_dict)
{1: 'Akhilaa', 2: [1, 12, 1988]}
```

2.4 DELETE OR REMOVE ELEMENTS FROM A DICTIONARY

- We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.
- The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method.
- We can also use the del keyword to remove individual items or the entire dictionary itself.
- **For example:**

I.
```
>>> sq = {1:2, 2:4, 3:9, 4:16, 5:25}
>>> print(sq)
{1: 2, 2: 4, 3: 9, 4: 16, 5: 25}
>>> print(sq.pop(3))
9
>>> print(sq)
{1: 2, 2: 4, 4: 16, 5: 25}
>>> print(sq.popitem())
(5, 25)
>>> print(sq)
{1: 2, 2: 4, 4: 16}
>>> del sq[2]
>>> print(sq)
{1: 2, 4: 16}
```

II.
```
>>> my_dict = {'name':'akhilaa', 1:[10,20,30]}
>>> print(my_dict.pop('name'))
akhilaa
>>> print(my_dict)
{1: [10, 20, 30]}
```

NOTE:

- The len function works on dictionaries; it returns the number of key-value pairs.

For example:

```
>>> my_dict = {'name':'akhilaa', 1:[10,20,30]}
```

```
>>> print(len(my_dict))
2
```

- The in operator works on dictionaries; it tells you whether something appears as a *key* in the dictionary.

For example:

```
>>> my_dict = {'name':'akhilaa', 1:[10,20,30]}
>>> 'name' in my_dict
True
>>> 'age' in my_dict
False
```

- To see whether something appears as a value in a dictionary, we can use the method values, which returns the values as a list, and then use the in operator.

For example:

```
>>> my_dict = {'name':'akhilaa', 1:[10,20,30]}
>>> values = list(my_dict.values())
>>> print(values)
['akhilaa', [10, 20, 30]]
>>> 10 in values
False
>>> 'akhilaa' in values
True
>>> [10,20,30] in values
True
>>> [10,30] in values
False
```

- The in operator uses different algorithms for lists and dictionaries. For lists, it uses a linear search algorithm. As the list gets longer, the search time gets longer in direct proportion to the length of the list. For dictionaries, Python uses an algorithm called a *hash table* that has a remarkable property: the in operator takes about the same amount of time no matter how many items there are in a dictionary.
- The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

For example:

```
>>> d = {'b':2, 'z':26, 'a':1}
>>> t = list(d.items())
>>> print(t)
[('b', 2), ('z', 26), ('a', 1)]
>>> t.sort()
>>> print(t)
[('a', 1), ('b', 2), ('z', 26)]
```

2.5 DICTIONARY AS A SET OF COUNTERS

- Suppose we are given a string and we want to count how many times each letter appears. There are several ways we could do it:
    1. We could create 26 variables, one for each letter of the alphabet. Then we could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional.
    2. We could create a list with 26 elements. Then we could convert each character to a number (using the built-in function ord), use the number as an index into the list, and increment the appropriate counter.
    3. We could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, we would add an item to the dictionary. After that you would increment the value of an existing item.
- Each of these options performs the same computation, but each of them implements that computation in a different way.
- An *implementation* is a way of performing a computation; some implementations are better than others. For example, an advantage of the dictionary implementation is that we don't have to know ahead of time which letters appear in the string and we only have to make room for the letters that do appear.
- **For example:**

```
#To find frequency of each character in a string.

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)
```

Output:

1. **Enter the string: Pseudopseudohypoparathyroidism**
   {'P': 1, 's': 3, 'e': 2, 'u': 2, 'd': 3, 'o': 4, 'p': 3, 'h': 2, 'y': 2, 'a': 2, 'r': 2, 't': 1, 'i': 2, 'm': 1}
2. **Enter the string: hi. my name is akhilaa. welcome to python world.**
   {'h': 3, 'i': 3, '.': 3, ' ': 8, 'm': 3, 'y': 2, 'n': 2, 'a': 4, 'e': 3, 's': 1, 'k': 1, 'l': 3, 'w': 2, 'c': 1, 'o': 4, 't': 2, 'p': 1, 'r': 1, 'd': 1}

The for loop traverses the string. Each time through the loop, if the character c is not in the dictionary, we create a new item with key c and the initial value 1 (since we have seen this letter once). If c is already in the dictionary we increment d[c].

- Dictionaries have a method called get that takes a key and a default value. If the key appears in the dictionary, get returns the corresponding value; otherwise it returns the default value.
- **For example:**

```
>>> d = {'akhilaa':27, 'arnitha':19, 'anusha':23}
>>> print(d.get('akhilaa',0))
27
>>> print(d.get('saanvi',0))
0
```

- Since the get method automatically handles the case where a key is not in a dictionary, we can reduce the previous program of four lines down to one and eliminate the if statement.
- **For example:**

```
#To find frequency of each character in a string.

word = input('Enter the string: ')
d = dict()
for c in word:
   d[c] = d.get(c,0) + 1
print(d)
```

Output:

1. **Enter the string: Pseudopseudohypoparathyroidism**
   {'P': 1, 's': 3, 'e': 2, 'u': 2, 'd': 3, 'o': 4, 'p': 3, 'h': 2, 'y': 2, 'a': 2, 'r': 2, 't': 1, 'i': 2, 'm': 1}
2. **Enter the string: hi. my name is akhilaa. welcome to python world.**
   {'h': 3, 'i': 3, '.': 3, ' ': 8, 'm': 3, 'y': 2, 'n': 2, 'a': 4, 'e': 3, 's': 1, 'k': 1, 'l': 3, 'w': 2, 'c': 1, 'o': 4, 't': 2, 'p': 1, 'r': 1, 'd': 1}

2.6 DICTIONARIES AND FILES

- One of the common uses of a dictionary is to count the occurrence of words in a file with some written text.
- We will write a Python program to read through the lines of the file, break each line into a list of words, and then loop through each of the words in the line and count each word using a dictionary.
- We have two for loops. The outer loop is reading the lines of the file and the inner loop is iterating through each of the words on that particular line. This is an example of a pattern called *nested loops* because one of the loops is the *outer* loop and the other loop is the *inner* loop.
- **For example:**

In poem.txt

```
When to the session of sweet silent thought
I summon up remembrance of things past
I sigh the lack of many a thing I sought
And with old woes new wail my dear time is waste
```

```
The sad account of fore-bemoaned moan
Which I new pay as if not paid before
But if the while I think on thee dear friend
All losses are restored and sorrows end
```

In dictword.py

```
#To find frequency of words in a file.

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)
```

Output:

```
Enter the file name: poem.txt
{'When': 1, 'to': 1, 'the': 3, 'session': 1, 'of': 4, 'sweet': 1, 'silent': 1, 'thought': 1, 'I': 5,
'summon': 1, 'up': 1, 'remembrance': 1, 'things': 1, 'past': 1, 'sigh': 1, 'lack': 1, 'many': 1,
'a': 1, 'thing': 1, 'sought': 1, 'And': 1, 'with': 1, 'old': 1, 'woes': 1, 'new': 2, 'wail': 1, 'my': 1,
'dear': 2, 'time': 1, 'is': 1, 'waste': 1, 'The': 1, 'sad': 1, 'account': 1, 'fore-bemoaned': 1,
'moan': 1, 'Which': 1, 'pay': 1, 'as': 1, 'if': 2, 'not': 1, 'paid': 1, 'before': 1, 'But': 1, 'while':
1, 'think': 1, 'on': 1, 'thee': 1, 'friend': 1, 'All': 1, 'losses': 1, 'are': 1, 'restored': 1, 'and': 1,
'sorrows': 1, 'end': 1}
```

2.7 LOOPING AND DICTIONARIES

- If we use a dictionary as the sequence in a for statement, it traverses the keys of the dictionary. This loop prints each key and the corresponding value.

For example:

```
names = {'akhilaa':1990, 'nikkhil':1988, 'chandra':1972}
for k in names:
   print(k, names[k])
```

Output:

```
akhilaa 1990
nikkhil 1988
chandra 1972
```

- If we wanted to find all the entries in a dictionary with a value above 1990, we could write the following code. The for loop iterates through the *keys* of the dictionary, so we must use the index operator to retrieve the corresponding *value* for each key.

For example:

```
names = {'akhilaa':1990, 'nikkhil':1988, 'chandra':1972}
for k in names:
   if names[k] > 1980:
      print(k, names[k])
```

Output:

```
akhilaa 1990
nikkhil 1988
```

- If we want to print the keys in alphabetical order, we first make a list of the keys in the dictionary using the keys method available in dictionary objects, and then sort that list and loop through the sorted list, looking up each key and printing out key-value pairs in sorted order as follows.

For example:

```
names = {'akhilaa':1990, 'nikkhil':1988, 'chandra':1972}
lst = list(names.keys())
print(lst)
lst.sort()
for k in lst:
   print(k, names[k])
```

Output:

```
['akhilaa', 'nikkhil', 'chandra']
akhilaa 1990
chandra 1972
nikkhil 1988
```

Note:

Python String translate() Method

- The method **translate()** returns a copy of the string in which all characters have been translated using *table* (constructed with the maketrans() function in the string module), optionally deleting all characters found in the string *deletechars*.
- The syntax is:

```
str.translate(table[, deletechars])
```

  - **table** – we can use the maketrans() helper function in the string module to create a translation table.
  - **deletechars** – The list of characters to be removed from the source string.
- This method returns a translated copy of the string.

For example:

```
intab = "aeiou"
outtab = "12345"

string = "welcome to python programming!!!"

res = string.translate(string.maketrans(intab, outtab))
print('After translation:',res)

res1 = string.translate(string.maketrans(intab, outtab, 'pyth'))
print('After deletion:',res1)
```

Output:

```
After translation: w2lc4m2 t4 pyth4n pr4gr1mm3ng!!!
After deletion: w2lc4m2 4 4n r4gr1mm3ng!!!
```

2.8 ADVANCED TEXT PARSING

- Consider the following contents of file romeo.txt:

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,

- Since the Python split function looks for spaces and treats words as tokens separated by spaces, we would treat the words “soft!” and “soft” as *different* words and create a separate dictionary entry for each word.
- Also since the file has capitalization, we would treat “who” and “Who” as different words with different counts.
- We can solve both these problems by using the string methods lower, punctuation, and translate. The translate is the most subtle of the methods. Here is the documentation for translate:

```
line.translate(str.maketrans(fromstr, tostr, deletestr))
```

Replace the characters in fromstr with the character in the same position in tostr and delete all characters that are in deletestr. The fromstr and tostr can be empty strings and the deletestr parameter can be omitted.

- We will not specify the table but we will use the deletechars parameter to delete all of the punctuation. We will even let Python tell us the list of characters that it considers "punctuation":

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Note: String.punctuation - String of ASCII characters which are considered punctuation characters

- **For example:**

```
#To find frequency of words by removing punctuations in a file.

import string

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

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

counts = dict()
for line in fhand:
   line = line.rstrip()
   line = line.translate(line.maketrans('', '', string.punctuation))
   line = line.lower()
   words = line.split()
   for word in words:
      if word not in counts:
         counts[word] = 1
      else:
         counts[word] += 1

print(counts)
```

Output:

```
Enter the file name: romeo.txt
```

{'but': 1, 'soft': 1, 'what': 1, 'light': 1, 'through': 1, 'yonder': 1, 'window': 1, 'breaks': 1, 'it': 1, 'is': 3, 'the': 3, 'east': 1, 'and': 3, 'juliet': 1, 'sun': 2, 'arise': 1, 'fair': 1, 'kill': 1, 'envious': 1, 'moon': 1, 'who': 1, 'already': 1, 'sick': 1, 'pale': 1, 'with': 1, 'grief': 1}

2.9 EXERCISES

1. Write a program to find frequency of words in a file.
2. Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.
3. Write a python program to add a key to a dictionary.
4. Write a python program to concatenate following dictionaries to create a new one.
5. Write a python program to check if a given key already exists in a dictionary.
6. Write a python program to iterate over dictionaries using for loops.
7. Write a python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).
8. Write a python script to merge two python dictionaries.
9. Write a python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys.
10. Write a python program to sum all the items in a dictionary.
11. Write a python program to remove a key from a dictionary.
12. Write a python program to map two lists into a dictionary.
13. Write a python program to sort a dictionary by key.

2.10 GLOSSARY

dictionary	A mapping from a set of keys to their corresponding values.
hashtable	The algorithm used to implement Python dictionaries.
hash function	A function used by a hashtable to compute the location for a key.
histogram	A set of counters.
implementation	A way of performing a computation.
item	Another name for a key-value pair.
key	An object that appears in a dictionary as the first part of a key-value pair.
key-value pair	The representation of the mapping from a key to a value.
lookup	A dictionary operation that takes a key and finds the corresponding value.
nested loops	When there are one or more loops “inside” of another loop. The inner loop runs to completion each time the outer loop runs once.
value	An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word “value”.

3 TUPLES

- **A tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.**

  OR

  A tuple is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are *immutable*.
- Some of the advantages of tuples over lists are:
  - We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
  - Since tuples are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
  - Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
  - If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

  For example:

```
>>> a = list(range(1000))
>>> b = tuple(range(1000))
>>> a.__sizeof__()
4544
>>> b.__sizeof__()
4012
```

- Tuples are also *comparable* and *hashable* so we can sort lists of them and use tuples as key values in Python dictionaries.

3.1 CREATING A TUPLE

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).
- **For example:**

  I.
```
>>> my_tuple = ()          #empty tuple.
>>> print(my_tuple)
```

```
()
```

II.
```
>>> my_tuple = (10, 20, 30)        #tuple having integers.
>>> print(my_tuple)
(10, 20, 30)
```

III.
```
>>> my_tuple = (12, 'akhilaa', -30.65)     #tuple having mixed data.
>>> print(my_tuple)
(12, 'akhilaa', -30.65)
```

IV.
```
>>> my_tuple = ('good', [34,51,13], (-25,14,0.245))      #nested tuple.
>>> print(my_tuple)
('good', [34, 51, 13], (-25, 14, 0.245))
```

V.
```
>>> my_tuple = 12, 'akhilaa'       #tuple can be created without using '( )'
>>> print(my_tuple)
(12, 'akhilaa')
```

VI.
```
>>> a, b = my_tuple
>>> print(a)
12
>>> print(b)
akhilaa
```

- To create a tuple with a single element, you have to include the final comma.

  For example:

```
>>> t = ('z',)
>>> type(t)
<class 'tuple'>
```

  Without the comma Python treats ('a') as an expression with a string in parentheses that evaluates to a string.

  For example:

```
>>> t = ('z')
>>> type(t)
<class 'str'>
```

- Another way to construct a tuple is the built-in function tuple. With no argument, it creates an empty tuple.

  For example:

```
>>> tup = tuple()
>>> print(tup)
()
```

- If the argument is a sequence (string, list, or tuple), the result of the call to tuple is a tuple with the elements of the sequence.

  For example:

I.
```
>>> t = tuple('python')
>>> print(t)
('p', 'y', 't', 'h', 'o', 'n')
```

II. >>> t = tuple([1,24,13,76,65])
>>> print(t)
(1, 24, 13, 76, 65)

III. >>> t = tuple('akhilaa',27.3, 12)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: tuple() takes at most 1 argument (3 given)

- Because tuple is the name of a constructor, you should avoid using it as a variable name.

3.2 ACCESSING ELEMENTS FROM A LIST

- There are various ways in which we can access the elements of a tuple:

1. **Indexing**
   - ✓ The index operator [] is used to access an item in a tuple where the index starts from 0. So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other that (6, 7,...) will raise an IndexError.
   - ✓ The index must be an integer, so we cannot use float or other types. This will result into TypeError.
   - ✓ The nested tuple are accessed using nested indexing.
   - ✓ **For example:**

1. >>> tup = ('p', 'y', 't', 'h', 'o', 'n')
>>> print(tup[0])
p
>>> print(tup[5])
n
>>> print(tup[7])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
>>> print(tup[1.4])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: tuple indices must be integers or slices, not float
2. >>> ntup = ('awesome', [13, -54, 0.24], (93, 8436, 37))
>>> print(ntup[0][3])
s
>>> print(ntup[2])
(93, 8436, 37)
>>> print(ntup[2][3])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: tuple index out of range

```
>>> print(ntup[2][1])
8436
>>> print(ntup[1][2])
0.24
```

3.
```
>>> ntup = ('awesome', [13, -54, 0.24], (93, 8436, 37),{'date':3, 'month':4, 'year':2018})
>>> print(ntup[3])
{'date': 3, 'month': 4, 'year': 2018}
>>> print(ntup[3][2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
>>> print(ntup[3]['month'])
4
```

2. Negative Indexing

- ✓ Python allows negative indexing for its sequences.
- ✓ The index of -1 refers to the last item, -2 to the second last item and so on.
- ✓ **For example:**

```
>>> print(ntup[-3])
[13, -54, 0.24]
>>> print(ntup[-3][2])
0.24
>>> print(ntup[-4][-1])
e
>>> print(ntup[-4][5])
m
>>> print(ntup[2][-3])
93
```

3. Slicing

- ✓ We can access a range of items in a tuple by using the slicing operator - colon ":".
- ✓ **For example:**

```
>>> ntup = ('awesome', [13, -54, 0.24], (93, 8436, 37),{'date':3, 'month':4, 'year':2018})
>>> print(ntup[:])
('awesome', [13, -54, 0.24], (93, 8436, 37), {'date': 3, 'month': 4, 'year': 2018})
>>> print(ntup[1:])
([13, -54, 0.24], (93, 8436, 37), {'date': 3, 'month': 4, 'year': 2018})
>>> print(ntup[:3])
('awesome', [13, -54, 0.24], (93, 8436, 37))
>>> print(ntup[0][1:5])
weso
>>> print(ntup[1][-1:-5])
```

```
[]
>>> print(ntup[1][-3:-1])
[13, -54]
>>> print(ntup[1][-1:-3])
[]
>>> print(ntup[-1]['date':])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'slice'
```

3.3 TUPLES ARE IMMUTABLE

- Tuples are immutable which means you cannot update or change the values of tuple elements.
- We can take portions of existing tuples to create new tuples.

For example:

```
>>> tup = (7, 86, 43)
>>> tup[0] = 78
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

3.4 DELETING ELEMENTS

- Removing individual tuple elements is not possible.

For example:

```
>>> tup = (7, 86, 43)
>>> del tup[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>> del tup
>>> print(tup)
Traceback (most recent call last):     # an exception raised, this is because after del
File "<stdin>", line 1, in <module>          # tup tuple does not exist any more
NameError: name 'tup' is not defined
```

3.5 TUPLES OPERATIONS

- Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

Python Expression	Results	Description

len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

3.6 COMPARING TUPLES

- The comparison operators work with tuples and other sequences.
- Python starts by comparing the first element from each sequence. If they are equal, it goes on to the next element, and so on, until it finds elements that differ. Subsequent elements are not considered (even if they are really big).

  For example:

  I. **>>> (0, 1, 2) < (0, 2, 3)**
     True
  II. **>>> (0, 1, 20000000) < (0, 2, 3)**
     True
  III. **>>> (10, 21, 20) < (0, 2, 3)**
     False
- 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.
- **For example:**

```
#Python program to sort a list of words from longest to shortest.

txt = 'The grass is always greener on the other side of the fence'

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)
```

Output:

```
The list is:
 [(3, 'The'), (5, 'grass'), (2, 'is'), (6, 'always'), (7, 'greener'), (2, 'on'), (3, 'the'), (5, 'other'), (4, 'side'), (2, 'of'), (3, 'the'), (5, 'fence')]

The list after sorting is:
 [(7, 'greener'), (6, 'always'), (5, 'other'), (5, 'grass'), (5, 'fence'), (4, 'side'), (3, 'the'), (3, 'the'), (3, 'The'), (2, 'on'), (2, 'of'), (2, 'is')]

The sorted list is:
 ['greener', 'always', 'other', 'grass', 'fence', 'side', 'the', 'the', 'The', 'on', 'of', 'is']
```

- The first loop builds a list of tuples, where each tuple is a word preceded by its length.
- sort compares the first element, length, first, and only considers the second element to break ties. The keyword argument reverse=True tells sort to go in decreasing order.
- The second loop traverses the list of tuples and builds a list of words in descending order of length.

3.7 TUPLE ASSIGNMENT

- One of the unique syntactic features of the Python language is the ability to have a tuple on the left side of an assignment statement. This allows you to assign more than one variable at a time when the left side is a sequence.
- **For example:**

```
>>> a = ['enjoy', 5]
>>> x, y = a
>>> x
'enjoy'
>>> y
5
```

- Python *roughly* translates the tuple assignment syntax to be the following:

```
>>> a = ['enjoy', 5]
>>> x = a[0]
>>> y = a[1]
```

```
>>> x
'enjoy'
>>> y
5
```

- When we use a tuple on the left side of the assignment statement, we omit the parentheses, but the following is an equally valid syntax:

```
>>> a = ['enjoy', 5]
>>> (x, y) = a
>>> x
'enjoy'
>>> y
5
```

- A tuple assignment allows us to *swap* the values of two variables in a single statement:

```
>>> a = 10
>>> b = 20
>>> a, b = b, a
>>> print(a)
20
>>> print(b)
10
```

Both sides of this statement are tuples, but the left side is a tuple of variables; the right side is a tuple of expressions. Each value on the right side is assigned to its respective variable on the left side. All the expressions on the right side are evaluated before any of the assignments.

- The number of variables on the left and the number of values on the right must be the same:

```
>>> a, b = 1, 2, 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

- The right side can be any kind of sequence (string, list, or tuple).

For example, to split an email address into a user name and a domain, we could write:

```
>>> addr = 'akhilaa@cmrit.ac.in'
>>> uname, dname = addr.split('@')
>>> print(uname)
akhilaa
>>> print(dname)
cmrit.ac.in
```

The return value from split is a list with two elements; the first element is assigned to uname, the second to dname.

3.8 DICTIONARIES AND TUPLES

- Dictionaries have a method called items that returns a list of tuples, where each tuple is a key-value pair:

```
>>> d = {'b':10, 'a': -78, 'c':5}
>>> t = list(d.items())
>>> print(t)
[('b', 10), ('a', -78), ('c', 5)]
```

- In dictionaries, items are not in a particular order. Since the list of tuples is a list, and tuples are comparable, we can now sort the list of tuples.

```
>>> d = {'b':10, 'a': -78, 'c':5}
>>> t = list(d.items())
>>> print(t)
[('b', 10), ('a', -78), ('c', 5)]
>>> t.sort()
>>> print(t)
[('a', -78), ('b', 10), ('c', 5)]
```

3.9 MULTIPLE ASSIGNMENTS WITH DICTIONARIES

- We can combine items, tuple assignment, and for, we can see a nice code pattern for traversing the keys and values of a dictionary in a single loop:

```
d = {'a':10, 'b':1, 'c':22}
t = list(d.items())
print(t)

for k, val in t:
   print(val, k)
```

Output:

```
[('a', 10), ('b', 1), ('c', 22)]
10 a
1 b
22 c
```

This loop has two *iteration variables* because items returns a list of tuples and key, val is a tuple assignment that successively iterates through each of the key-value pairs in the dictionary.

For each iteration through the loop, both key and value are advanced to the next key-value pair in the dictionary (still in hash order).

- **# Python program to print the contents of a dictionary sorted by the value stored in each key-value pair.**

```
d = {'a':10, 'b':1, 'c':22}
```

```
l = list()

for key, val in d.items() :
   l.append( (val, key) )

print('The list before sorting:',l)

l.sort(reverse=True)

print('The list after sorting:',l)
```

Output:

The list before sorting: [(10, 'a'), (1, 'b'), (22, 'c')]

The list after sorting: [(22, 'c'), (10, 'a'), (1, 'b')]

First make a list of tuples where each tuple is (value, key). The items method would give us a list of (key, value) tuples, but this time we want to sort by value, not key. Once we have constructed the list with the value-key tuples, it is a simple matter to sort the list in reverse order and print out the new, sorted list.

3.10 PROGRAM TO FIND MOST COMMON WORDS

- The first part of the program which reads the file and computes the dictionary that maps each word to the count of words in the document is unchanged. But instead of simply printing out counts and ending the program, we construct a list of (val, key) tuples and then sort the list in reverse order.
- Since the value is first, it will be used for the comparisons. If there is more than one tuple with the same value, it will look at the second element (the key), so tuples where the value is the same will be further sorted by the alphabetical order of the key.
- At the end we write a nice for loop which does a multiple assignment iteration and prints out the ten most common words by iterating through a slice of the list (lst[:10]).
- **In file romeo.txt**

```
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,
```

In file tup.py

```
#Python program to find the most common words in a file.

import string

fhand = open('romeo.txt')
```

```
counts = dict()

for line in fhand:
   line = line.translate(str.maketrans('', '', string.punctuation))
   line = line.lower()
   words = line.split()
   for word in words:
      if word not in counts:
         counts[word] = 1
      else:
         counts[word] += 1

#Sorts the dictionary by value
lst = list()
for key, val in list(counts.items()):
   lst.append((val, key))

lst.sort(reverse=True)

for key, val in lst[:10]:
   print(key, val)
```

Output:

```
3 the
3 is
3 and
2 sun
1 yonder
1 with
1 window
1 who
1 what
1 through
```

3.11 USING TUPLES AS KEYS IN DICTIONARIES

- Because tuples are *hashable* and lists are not, if we want to create a *composite* key to use in a dictionary we must use a tuple as the key.
- We would encounter a composite key if we wanted to create a telephone directory that maps from last-name, first-name pairs to telephone numbers. Assuming that we have defined the variables last, first, and number, we could write a dictionary assignment statement as follows:

```
directory[last,first] = number
```

- The expression in brackets is a tuple. We could use tuple assignment in a for loop to traverse this dictionary.

```
for last, first in directory:
    print(first, last, directory[last,first])
```

This loop traverses the keys in directory, which are tuples. It assigns the elements of each tuple to last and first, then prints the name and corresponding telephone number.

3.12 SEQUENCES: STRINGS, LISTS, AND TUPLES

- Strings are more limited than other sequences because the elements have to be characters. They are also immutable. If we need the ability to change the characters in a string (as opposed to creating a new string), we might want to use a list of characters instead.
- Lists are more common than tuples, mostly because they are mutable. But there are a few cases where you might prefer tuples:
    1. In some contexts, like a return statement, it is syntactically simpler to create a tuple than a list. In other contexts, you might prefer a list.
    2. If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.
    3. If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behaviour due to aliasing.
- Because tuples are immutable, they don't provide methods like sort and reverse, which modify existing lists. However Python provides the built-in functions sorted and reversed, which take any sequence as a parameter and return a new sequence with the same elements in a different order.

3.13 DIFFERENCE BETWEEN LIST AND TUPLE

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

3.14 EXERCISES

1. Write a program that reads a file and prints the letters in decreasing order of frequency. Your program should convert all the input to lower case and only count the letters a-z. Your program

should not count spaces, digits, punctuation, or anything other than the letters a-z. Find text samples from several different languages and see how letter frequency varies between languages.

3.15 GLOSSARY

comparable	A type where one value can be checked to see if it is greater than, less than, or equal to another value of the same type. Types which are comparable can be put in a list and sorted.
data structure	A collection of related values, often organized in lists, dictionaries, tuples, etc.
DSU	Abbreviation of "decorate-sort-undecorate", a pattern that involves building a list of tuples, sorting, and extracting part of the result.
gather	The operation of assembling a variable-length argument tuple.
hashable	A type that has a hash function. Immutable types like integers, floats, and strings are hashable; mutable types like lists and dictionaries are not.
scatter	The operation of treating a sequence as a list of arguments.
shape (of a data structure)	A summary of the type, size, and composition of a data structure.
singleton	A list (or other sequence) with a single element.
tuple	An immutable sequence of elements.
tuple assignment	An assignment with a sequence on the right side and a tuple of variables on the left. The right side is evaluated and then its elements are assigned to the variables on the left.

4 REGULAR EXPRESSIONS

- Regular expressions are a powerful language for matching text patterns.
- It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.
- While using the regular expression the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string.
- The regular expression library re must be imported into your program before we can use it.

  https://docs.python.org/3/library/re.html
- The simplest use of the regular expression library is the search() function. The following program demonstrates a trivial use of the search function.

  For example:

  In mbox.txt

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.90])
          by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
          Sat, 05 Jan 2008 09:14:16 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
          by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
          Sat, 05 Jan 2008 09:14:16 -0500
Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
          by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
          Sat, 5 Jan 2008 09:14:15 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
          BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
          5 Jan 2008 09:14:10 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
          by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
          Sat,  5 Jan 2008 14:10:05 +0000 (GMT)
Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
        by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 899
        for <source@collab.sakaiproject.org>;
        Sat, 5 Jan 2008 14:09:50 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
          by shmi.uhi.ac.uk (Postfix) with ESMTP id A215243002
```

for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 14:13:33 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJVp010329
for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 09:12:19 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11/Submit) id m05ECIaH010327
for source@collab.sakaiproject.org; Sat, 5 Jan 2008 09:12:18 -0500
Date: Sat, 5 Jan 2008 09:12:18 -0500
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to stephen.marquard@uct.ac.za using -f
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/sakai_2-5-x/content-impl/impl/src/java/org/sakaiproject/content/impl
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Sat Jan 5 09:14:16 2008
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772

Author: stephen.marquard@uct.ac.za
Date: 2008-01-05 09:12:07 -0500 (Sat, 05 Jan 2008)
New Revision: 39772

Modified:
content/branches/sakai_2-5-x/content-impl/impl/src/java/org/sakaiproject/content/impl/ContentServiceSqlOracle.java
content/branches/sakai_2-5-x/content-impl/impl/src/java/org/sakaiproject/content/impl/DbContentService.java
Log:
SAK-12501 merge to 2-5-x: r39622, r39624:5, r39632:3 (resolve conflict from differing linebreaks for r39622)

This automatic notification message was sent by Sakai Collab (https://collab.sakaiproject.org/portal) from the Source site.
You can modify how you receive notifications at My Workspace > Preferences.

From louis@media.berkeley.edu Fri Jan 4 18:10:48 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.97])
by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
Fri, 04 Jan 2008 18:10:48 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
Fri, 04 Jan 2008 18:10:48 -0500
Received: from icestorm.mr.itd.umich.edu (icestorm.mr.itd.umich.edu [141.211.93.149])
by sleepers.mail.umich.edu () with ESMTP id m04NAbGa029441;
Fri, 4 Jan 2008 18:10:37 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
BY icestorm.mr.itd.umich.edu ID 477EBCE3.161BB.4320 ;
4 Jan 2008 18:10:31 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
by paploo.uhi.ac.uk (Postfix) with ESMTP id 07969BB706;
Fri, 4 Jan 2008 23:10:33 +0000 (GMT)
Message-ID: <200801042308.m04N8v6O008125@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 710
for <source@collab.sakaiproject.org>;
Fri, 4 Jan 2008 23:10:10 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
by shmi.uhi.ac.uk (Postfix) with ESMTP id 4BA2F42F57
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 23:10:10 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m04N8vHG008127
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 18:08:57 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11/Submit) id m04N8v6O008125
for source@collab.sakaiproject.org; Fri, 4 Jan 2008 18:08:57 -0500
Date: Fri, 4 Jan 2008 18:08:57 -0500
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to louis@media.berkeley.edu using -f
To: source@collab.sakaiproject.org

From: louis@media.berkeley.edu
Subject: [sakai] svn commit: r39771 - in bspace/site-manage/sakai_2-4-x/site-manage-tool/tool/src: bundle java/org/sakaiproject/site/tool
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Fri Jan 4 18:10:48 2008
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39771

Author: louis@media.berkeley.edu
Date: 2008-01-04 18:08:50 -0500 (Fri, 04 Jan 2008)
New Revision: 39771

Modified:
bspace/site-manage/sakai_2-4-x/site-manage-tool/tool/src/bundle/sitesetupgeneric.properties
bspace/site-manage/sakai_2-4-x/site-manage-tool/tool/src/java/org/sakaiproject/site/tool/SiteAction.java
Log:
BSP-1415 New (Guest) user Notification

This automatic notification message was sent by Sakai Collab (https://collab.sakaiproject.org/portal) from the Source site.
You can modify how you receive notifications at My Workspace > Preferences.

From zqian@umich.edu Fri Jan 4 16:10:39 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.25])
by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
Fri, 04 Jan 2008 16:10:39 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
Fri, 04 Jan 2008 16:10:39 -0500

Received: from ghostbusters.mr.itd.umich.edu (ghostbusters.mr.itd.umich.edu [141.211.93.144])
by panther.mail.umich.edu () with ESMTP id m04LAcZw014275;
Fri, 4 Jan 2008 16:10:38 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
BY ghostbusters.mr.itd.umich.edu ID 477EA0C6.A0214.25480 ;
4 Jan 2008 16:10:33 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
by paploo.uhi.ac.uk (Postfix) with ESMTP id C48CDBB490;
Fri, 4 Jan 2008 21:10:31 +0000 (GMT)
Message-ID: <200801042109.m04L92hb007923@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 906
for <source@collab.sakaiproject.org>;
Fri, 4 Jan 2008 21:10:18 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
by shmi.uhi.ac.uk (Postfix) with ESMTP id 7D13042F71
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 21:10:14 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m04L927E007925
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 16:09:02 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11/Submit) id m04L92hb007923
for source@collab.sakaiproject.org; Fri, 4 Jan 2008 16:09:02 -0500
Date: Fri, 4 Jan 2008 16:09:02 -0500
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to zqian@umich.edu using -f
To: source@collab.sakaiproject.org
From: zqian@umich.edu
Subject: [sakai] svn commit: r39770 - site-manage/branches/sakai_2-5-x/site-manage-tool/tool/src/webapp/vm/sitesetup
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Fri Jan 4 16:10:39 2008
X-DSPAM-Confidence: 0.6961
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39770

Author: zqian@umich.edu
Date: 2008-01-04 16:09:01 -0500 (Fri, 04 Jan 2008)
New Revision: 39770

Modified:
site-manage/branches/sakai_2-5-x/site-manage-tool/tool/src/webapp/vm/sitesetup/chef_site-siteInfo-list.vm
Log:
merge fix to SAK-9996 into 2-5-x branch: svn merge -r 39687:39688 https://source.sakaiproject.org/svn/site-manage/trunk/

This automatic notification message was sent by Sakai Collab (https://collab.sakaiproject.org/portal) from the Source site.
You can modify how you receive notifications at My Workspace > Preferences.

From rjlowe@iupui.edu Fri Jan 4 15:46:24 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.25])
by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
Fri, 04 Jan 2008 15:46:24 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
Fri, 04 Jan 2008 15:46:24 -0500
Received: from dreamcatcher.mr.itd.umich.edu (dreamcatcher.mr.itd.umich.edu [141.211.14.43])
by panther.mail.umich.edu () with ESMTP id m04KkNbx032077;
Fri, 4 Jan 2008 15:46:23 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
BY dreamcatcher.mr.itd.umich.edu ID 477E9B13.2F3BC.22965 ;
4 Jan 2008 15:46:13 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
by paploo.uhi.ac.uk (Postfix) with ESMTP id 4AE03BB552;
Fri, 4 Jan 2008 20:46:13 +0000 (GMT)
Message-ID: <200801042044.m04Kiem3007881@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit

Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 38
for <source@collab.sakaiproject.org>;
Fri, 4 Jan 2008 20:45:56 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
by shmi.uhi.ac.uk (Postfix) with ESMTP id A55D242F57
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 20:45:52 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m04KieqE007883
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 15:44:40 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11/Submit) id m04Kiem3007881
for source@collab.sakaiproject.org; Fri, 4 Jan 2008 15:44:40 -0500
Date: Fri, 4 Jan 2008 15:44:40 -0500
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to rjlowe@iupui.edu using -f
To: source@collab.sakaiproject.org
From: rjlowe@iupui.edu
Subject: [sakai] svn commit: r39769 - in gradebook/trunk/app/ui/src: java/org/sakaiproject/tool/gradebook/ui/helpers/beans java/org/sakaiproject/tool/gradebook/ui/helpers/producers webapp/WEB-INF webapp/WEB-INF/bundle
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Fri Jan 4 15:46:24 2008
X-DSPAM-Confidence: 0.7565
X-DSPAM-Probability: 0.0000

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39769

Author: rjlowe@iupui.edu
Date: 2008-01-04 15:44:39 -0500 (Fri, 04 Jan 2008)
New Revision: 39769

Modified:
gradebook/trunk/app/ui/src/java/org/sakaiproject/tool/gradebook/ui/helpers/beans/AssignmentGradeRecordBean.java
gradebook/trunk/app/ui/src/java/org/sakaiproject/tool/gradebook/ui/helpers/producers/GradeGradebookItemProducer.java
gradebook/trunk/app/ui/src/webapp/WEB-INF/applicationContext.xml

gradebook/trunk/app/ui/src/webapp/WEB-INF/bundle/messages.properties
gradebook/trunk/app/ui/src/webapp/WEB-INF/requestContext.xml
Log:
SAK-12180 - Fixed errors with grading helper

---------- This automatic notification message was sent by Sakai Collab (https://collab.sakaiproject.org/portal) from the Source site.
You can modify how you receive notifications at My Workspace > Preferences.

From zqian@umich.edu Fri Jan 4 15:03:18 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.46])
by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
Fri, 04 Jan 2008 15:03:18 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
Fri, 04 Jan 2008 15:03:18 -0500
Received: from firestarter.mr.itd.umich.edu (firestarter.mr.itd.umich.edu [141.211.14.83])
by fan.mail.umich.edu () with ESMTP id m04K3HGF006563;
Fri, 4 Jan 2008 15:03:17 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
BY firestarter.mr.itd.umich.edu ID 477E9100.8F7F4.1590 ;
4 Jan 2008 15:03:15 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
by paploo.uhi.ac.uk (Postfix) with ESMTP id 57770BB477;
Fri, 4 Jan 2008 20:03:09 +0000 (GMT)
Message-ID: <200801042001.m04K1cO0007738@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 622
for <source@collab.sakaiproject.org>;
Fri, 4 Jan 2008 20:02:46 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
by shmi.uhi.ac.uk (Postfix) with ESMTP id AB4D042F4D
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 20:02:50 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])

by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m04K1cXv007740
for <source@collab.sakaiproject.org>; Fri, 4 Jan 2008 15:01:38 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11/Submit) id m04K1cO0007738
for source@collab.sakaiproject.org; Fri, 4 Jan 2008 15:01:38 -0500
Date: Fri, 4 Jan 2008 15:01:38 -0500
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to zqian@umich.edu using -f
To: source@collab.sakaiproject.org
From: zqian@umich.edu
Subject: [sakai] svn commit: r39766 - site-manage/branches/sakai_2-4-x/site-manage-tool/tool/src/java/org/sakaiproject/site/tool
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Fri Jan 4 15:03:18 2008
X-DSPAM-Confidence: 0.7626
X-DSPAM-Probability: 0.0000

In regex.py

```
#Python program to search for lines that contain 'From:'

import re

hand = open('mbox.txt')

for line in hand:
   line = line.strip()
   if re.search('From:', line):
      print(line)
```

Output:

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu

- We open the file, loop through each line, and use the regular expression **search() to only print lines that contain the string "From:".** This program does not use the real power of regular expressions, since we could have used line.find() to accomplish the same result.
- The power of the regular expressions comes when we add special characters to the search string that allow us to more precisely control which lines match the string. Adding these special characters to our regular expression allow us to do sophisticated matching and extraction while writing very little code.
- For example, **the caret character is used in regular expressions to match "the beginning" of a line.** We could change our program to only match lines where "From:" was at the beginning of the line as follows:

#Python program to search for lines that contain 'From:' using regular expression.

```
import re

hand = open('mbox.txt')

for line in hand:
   line = line.strip()
   if re.search('^From:', line):
      print(line)
```

Output:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
```

We could have done equivalently with the startswith() method from the string library.

4.1 CHARACTER MATCHING IN REGULAR EXPRESSIONS

- **The most commonly used special character is the period or full stop, which matches any character.**
- In the following example, the regular expression "F..m:" would match any of the strings "From:", "Fxxm:", "F12m:", or "F!@m:" since the period characters in the regular expression match any character.

#Python program to search for lines that start with 'F', followed by 2 characters, followed by 'm:'

```
import re

hand = open('mbox.txt')
```

```
for line in hand:
    line = line.strip()
    if re.search('^F..m:', line):
        print(line)
```

Output:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
```

- This is particularly powerful when combined with the ability to indicate that **a character can be repeated any number of times using the "*" or "+" characters in your regular expression**. **These special characters mean that instead of matching a single character in the search string, they match zero-or-more characters (in the case of the asterisk) or one-or-more of the characters (in the case of the plus sign).**

```
# Python program to search for lines that start with From and have an at sign.

import re

hand = open('mbox.txt')

for line in hand:
    line = line.strip()
    if re.search('^From:.+@', line):
        print(line)
```

Output:

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
```

- The search string "^From:.+@" will successfully match lines that start with "From:", followed by one or more characters (".+"), followed by an at-sign
- We can think of the ".+" wildcard as expanding to **match all the characters between the colon character and the at-sign.**
- It is good to think of the plus and asterisk characters as "pushy".

4.2 EXTRACTING DATA USING REGULAR EXPRESSIONS

- If we want to extract data from a string in Python we can use the **findall() method to extract all of the substrings which match a regular expression.**
- This following program uses findall() to find the lines with email addresses in them and extract one or more addresses from each of those lines. **The findall() method searches the string in the second argument and returns a list of all of the strings that look like email addresses.** We are using **a two-character sequence that matches a non-whitespace character (\S).**

For example:

```
import re

s = 'A message from hod.ise@cmrit.ac.in to akhilaa@cmrit.ac.in about meeting @2PM'

lst = re.findall('\S+@\S+', s)
        OR
lst = re.findall('\S+@\S*', s)

print(lst)
```

Output:

```
['hod.ise@cmrit.ac.in', 'akhilaa@cmrit.ac.in']
```

- **Translating the regular expression, we are looking for substrings that have at least one non-whitespace character, followed by an at-sign, followed by at least one more non-whitespace character. The "\S+" matches as many non-whitespace characters as possible. The regular expression would match twice (hod.ise@cmrit.ac.in, akhilaa@cmrit.ac.in), but it would not match the string "@2PM" because there are no non-blank characters *before* the at-sign.**
- **Examples:**

1. **import re**

```
s = 'A message from hod.ise@cmrit.ac.in to akhilaa@cmrit.ac.in about meeting @2PM'

lst = re.findall('\S*@\S+', s)
        OR
lst = re.findall('\S*@\S*', s)

print(lst)
```

Output:

```
['hod.ise@cmrit.ac.in', 'akhilaa@cmrit.ac.in', '@2PM']
```

2. **import re**

   s = 'A message from hod.ise@cmrit.ac.in to akhilaa@cmrit.ac.in about meeting @2PM'

   lst = re.findall('\S@\S', s)

   print(lst)

   Output:

   ['e@c', 'a@c']

- We can use this regular expression in a program to read all the lines in a file and print out anything that looks like an email address as follows:

```
#Python program to search for lines that have an at sign between characters.

import re

hand = open('mbox.txt')

for line in hand:
   line = line.strip()
   x = re.findall('\S+@\S+', line)
   if len(x) > 0:
      print(x)
```

Output:

```
['stephen.marquard@uct.ac.za']
['<postmaster@collab.sakaiproject.org>']
['<200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['stephen.marquard@uct.ac.za']
['source@collab.sakaiproject.org']
['stephen.marquard@uct.ac.za']
['stephen.marquard@uct.ac.za']
['louis@media.berkeley.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801042308.m04N8v6O008125@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
```

['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['louis@media.berkeley.edu']
['source@collab.sakaiproject.org']
['louis@media.berkeley.edu']
['louis@media.berkeley.edu']
['zqian@umich.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801042109.m04L92hb007923@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['zqian@umich.edu']
['source@collab.sakaiproject.org']
['zqian@umich.edu']
['zqian@umich.edu']
['rjlowe@iupui.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801042044.m04Kiem3007881@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['rjlowe@iupui.edu']
['source@collab.sakaiproject.org']
['rjlowe@iupui.edu']
['rjlowe@iupui.edu']
['zqian@umich.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801042001.m04K1cO0007738@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
['zqian@umich.edu']
['source@collab.sakaiproject.org']

['zqian@umich.edu']

We read each line and then extract all the substrings that match our regular expression. Since findall() returns a list, we simply check if the number of elements in our returned list is more than zero to print only lines where we found at least one substring that looks like an email address.

- Some of our email addresses have incorrect characters like "<" or ";" at the beginning or end. Let's declare that we are only interested in the portion of the string that starts and ends with a letter or a number. To do this, we use another feature of regular expressions.
- Square brackets are used to indicate a set of multiple acceptable characters we are willing to consider matching. In a sense, the "\S" is asking to match the set of "non-whitespace characters".
- A new regular expression:

  [a-zA-Z0-9]\S*@\S*[a-zA-Z]

  Translating this regular expression, we are looking for substrings that start with a *single* lowercase letter, uppercase letter, or number "[a-zA-Z0-9]", followed by zero or more non-blank characters ("\S*"), followed by an at-sign, followed by zero or more non-blank characters ("\S*"), followed by an uppercase or lowercase letter. Note that we switched from "+" to "*" to indicate zero or more non-blank characters since "[a-zA-Z0-9]" is already one non-blank character. Remember that the "*" or "+" applies to the single character immediately to the left of the plus or asterisk.
- **For example:**

```
#Python program to search for lines that have an at sign between characters. The characters must be a letter or number.

import re

hand = open('mbox.txt')

for line in hand:
  line = line.strip()
  x = re.findall('[a-zA-Z0-9]\S+@\S+[a-zA-Z]', line)
  if len(x) > 0:
     print(x)
```

Output:

```
['stephen.marquard@uct.ac.za']
['postmaster@collab.sakaiproject.org']
['200801051412.m05ECIaH010327@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
['source@collab.sakaiproject.org']
['stephen.marquard@uct.ac.za']
```

['source@collab.sakaiproject.org']
['stephen.marquard@uct.ac.za']
['stephen.marquard@uct.ac.za']
['louis@media.berkeley.edu']
['postmaster@collab.sakaiproject.org']
['200801042308.m04N8v6O008125@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
['source@collab.sakaiproject.org']
['louis@media.berkeley.edu']
['source@collab.sakaiproject.org']
['louis@media.berkeley.edu']
['louis@media.berkeley.edu']
['zqian@umich.edu']
['postmaster@collab.sakaiproject.org']
['200801042109.m04L92hb007923@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
['source@collab.sakaiproject.org']
['zqian@umich.edu']
['source@collab.sakaiproject.org']
['zqian@umich.edu']
['zqian@umich.edu']
['rjlowe@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801042044.m04Kiem3007881@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
['source@collab.sakaiproject.org']
['rjlowe@iupui.edu']
['source@collab.sakaiproject.org']
['rjlowe@iupui.edu']
['rjlowe@iupui.edu']
['zqian@umich.edu']
['postmaster@collab.sakaiproject.org']
['200801042001.m04K1cO0007738@nakamura.uits.iupui.edu']

```
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
['source@collab.sakaiproject.org']
['zqian@umich.edu']
['source@collab.sakaiproject.org']
['zqian@umich.edu']
```

Notice that on the "source@collab.sakaiproject.org" lines, our regular expression eliminated two letters at the end of the string (">;"). This is because when we append "[a-zA-Z]" to the end of our regular expression, we are demanding that whatever string the regular expression parser finds must end with a letter. So when it sees the ">" after "sakaiproject.org>;" it simply stops at the last "matching" letter it found (i.e., the "g" was the last good match).

4.3 COMBINING SEARCHING AND EXTRACTING

- We want to find numbers on lines that start with the string "X-" such as:

  X-DSPAM-Confidence: 0.8475

  X-DSPAM-Probability: 0.0000
- We don't just want any floating-point numbers from any lines. We only want to extract numbers from lines that have the above syntax.
- We can construct the following regular expression to select the lines:

  ^X-.*: [0-9.]+

Translating this, we are saying, we want lines that start with "X-", followed by zero or more characters (".*"), followed by a colon (":") and then a space. After the space we are looking for one or more characters that are either a digit (0-9) or a period "[0-9.]+". Note that inside the square brackets, the period matches an actual period (i.e., it is not a wildcard between the square brackets).

For example:

```
# Python program to search for lines that start with 'X' followed by any non-whitespace characters and ':' followed by a space and any number. The number can include a decimal.

import re

hand = open('mbox.txt')

for line in hand:
   line = line.strip()
   if re.search('^X\S*: [0-9.]+', line):
      print(line)
```

Output:

```
X-DSPAM-Confidence: 0.8475
```

X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6961
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.7565
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.7626
X-DSPAM-Probability: 0.0000

- **Parentheses are another special character in regular expressions. When we add parentheses to a regular expression, they are ignored when matching the string. But when we are using findall(), parentheses indicate that while we want the whole expression to match, we only are interested in extracting a portion of the substring that matches the regular expression.**

#Python program to search for lines that start with 'X' followed by any non-whitespace characters and ':' followed by a space and any number. The number can include a decimal. Then print the number if it is greater than zero.

```
import re

hand = open('mbox.txt')

for line in hand:
   line = line.rstrip()
   x = re.findall('^X\S*: ([0-9.]+)', line)
   if len(x) > 0:
      print(x)
```

Output:

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
['0.7565']
['0.0000']
['0.7626']
['0.0000']
```

Instead of calling search(), we add parentheses around the part of the regular expression that represents the floating-point number to indicate we only want findall() to give us back the floating-point number portion of the matching string.

- As another example of this technique, if you look at the file there are a number of lines of the form:

  Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772

  If we wanted to extract all of the revision numbers (the integer number at the end of these lines) using the same technique as above, we could write the following program:

```
#Python program to search for lines that start with 'Details: rev=' followed by numbers and '.' Then print the number if it is greater than zero

import re

hand = open('mbox.txt')

for line in hand:
   line = line.strip()
   x = re.findall('^Details:.*rev=([0-9.]+)', line)
   if len(x) > 0:
      print(x)
```

  Output:

```
['39772']
['39771']
['39770']
['39769']
```

  Translating our regular expression, we are looking for lines that start with "Details:", followed by any number of characters (".*"), followed by "rev=", and then by one or more digits. We want to find lines that match the entire expression but we only want to extract the integer number at the end of the line, so we surround "[0-9]+" with parentheses.

- Now we can use regular expressions to redo an exercise where we were interested in the time of day of each mail message. We looked for lines of the form:

  From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

  and wanted to extract the hour of the day for each line. Previously we did this with two calls to split. First the line was split into words and then we pulled out the fifth word and split it again on the colon character to pull out the two characters we were interested in.

  While this worked, it actually results in pretty brittle code that is assuming the lines are nicely formatted. If you were to add enough error checking (or a big try/except block) to insure that your program never failed when presented with incorrectly formatted lines, the code would balloon to 10-15 lines of code that was pretty hard to read.

- We can do this in a far simpler way with the following regular expression:

  ^From .* [0-9][0-9]:

- The translation of this regular expression is that we are looking for lines that start with "From" (note the space), followed by any number of characters (".*"), followed by a space, followed by two digits "[0-9][0-9]", followed by a colon character. This is the definition of the kinds of lines we are looking for.

- In order to pull out only the hour using findall(), we add parentheses around the two digits as follows:

  ^From .* ([0-9][0-9]):

- **#Python program to search for lines that start with From and a character followed by a two digit number between 00 and 99 followed by ':'. Then print the number if it is greater than zero**

```
import re

hand = open('mbox-short.txt')

for line in hand:
   line = line.strip()
   x = re.findall('^From .* ([0-9][0-9]):', line)
   if len(x) > 0:
      print(x)
```

Output:

```
['09']
['18']
['16']
['15']
['15']
```

4.4 ESCAPE CHARACTER

- Since we use special characters in regular expressions to match the beginning or end of a line or specify wild cards, we need a way to indicate that these characters are “normal” and we want to match the actual character such as a dollar sign or caret.
- We can indicate that we want to simply match a character by prefixing that character with a backslash. For example, we can find money amounts with the following regular expression.

```
import re

x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

- Since we prefix the dollar sign with a backslash, it actually matches the dollar sign in the input string instead of matching the “end of line”, and the rest of the regular expression matches one or more digits or the period character. *Note:* Inside square brackets, characters are not “special”. So when we say “[0-9.]”, it really means digits or a period. Outside of square brackets, a period is the “wildcard” character and matches any character. Inside square brackets, the period is a period.

4.5 SUMMARY

^	Matches the beginning of the line.
$	Matches the end of the line.
.	Matches any character (a wildcard).
\s	Matches a whitespace character.
\S	Matches a non-whitespace character (opposite of \s).
*****	Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s).
***?**	Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s) in “non-greedy mode”.
+	Applies to the immediately preceding character and indicates to match one or more of the preceding character(s).
+?	Applies to the immediately preceding character and indicates to match one or more of the preceding character(s) in “non-greedy mode”.
[aeiou]	Matches a single character as long as that character is in the specified set. In this example, it would match “a”, “e”, “i”, “o”, or “u”, but no other characters.
[a-z0-9]	You can specify ranges of characters using the minus sign. This example is a single character that must be a lowercase letter or a digit.
[ˆA-Za-z]	When the first character in the set notation is a caret, it inverts the logic. This example matches a single character that is anything *other than* an uppercase or lowercase letter.
( )	When parentheses are added to a regular expression, they are ignored for the purpose of matching, but allow you to extract a particular subset of the matched string rather than the whole string when using findall().
\b	Matches the empty string, but only at the start or end of a word.
\B	Matches the empty string, but not at the start or end of a word.
\d	Matches any decimal digit; equivalent to the set [0-9].
\D	Matches any non-digit character; equivalent to the set [ˆ0-9].
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
{n}	Matches exactly n number of occurrences of preceding expression.
{ n,}	Matches n or more occurrences of preceding expression.
{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
\w	Matches word characters.
\W	Matches non-word characters.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.

4.6 EXERCISES

1. Write a python program to validate a phone number.
   ^+91-[9876]\d{9}$
   OR
   +91-[9876]\d{9}$
2. Write a python program to validate a PAN.
   ^[A-Z]{5}\d{4}[A-Z]$
   OR
   ^[A-Z]{5}[0-9]{4}[A-z]$
3. Write a python program to validate USN.
   [1-9]\S[A-Z]{2}\S[0-9]{2}\S[A-Z]{2}\S[0-9]{3}
4. Write a python program to extract floating point values from a file and store in a list.
5. Write a python program that matches a string that has an *a* followed by zero or more b's.
6. Write a python program to find sequences of lowercase letters joined with a underscore.
7. Write a python program that matches a string that has an 'a' followed by anything, ending in 'b'.
8. Write a python program to match a string that contains only upper and lowercase letters, numbers, and underscores.
9. Write a python program to separate and print the numbers of a given string.
10. Write a python program to find all words starting with 'a' or 'e' in a given string.

4.7 GLOSSARY

brittle code	Code that works when the input data is in a particular format but is prone to breakage if there is some deviation from the correct format. We call this “brittle code” because it is easily broken.
greedy matching	The notion that the “+” and “*” characters in a regular expression expand outward to match the largest possible string.
grep	A command available in most Unix systems that searches through text files looking for lines that match regular expressions. The command name stands for “Generalized Regular Expression Parser”.
regular expression	A language for expressing more complex search strings. A regular expression may contain special characters that indicate that a search only matches at the beginning or end of a line or many other similar capabilities.
wild card	A special character that matches any character. In regular expressions the wild-card character is the period.

QUESTIONS

1. What is a list? Explain how to create and access a list of elements with examples.
2. Are lists immutable or mutable? Explain.
3. With examples explain the operations that can be performed on lists.

4. Explain list slicing with examples.
5. Explain the following list functions with examples:
    a. append()
    b. extend()
    c. sort()
    d. strip()
    e. insert()
    f. reverse()
    g. index()
    h. count()
6. Explain the following list functions with examples:
    a. pop()
    b. del
    c. remove()
7. Explain the following list functions with examples:
    a. len()
    b. min()
    c. max()
    d. sum()
8. Explain split() and join() functions with examples.
9. What is a dictionary? Explain how to create and access elements in a dictionary with examples.
10. Are dictionaries immutable or mutable? Explain.
11. Explain pop() and popitem() functions with examples.
12. How can you delete an element from a dictionary? Explain with an example for each.
13. Explain the use of get() function.
14. Explain values() and items() functions on dictionary.
15. Explain maketrans() and translate() functions with examples.
16. Write a short note on advanced text parsing.
17. What is a tuple? Explain how to create and access elements of a tuple with examples.
18. Are tuples immutable or mutable? Explain.
19. What is DSU?
20. Explain tuple assignment with an example.
21. Difference between a list and a tuple.
22. What is regular expression? Explain how data is extracted using regular expressions with example.
23. Explain search() and findall() functions with examples.