:Student Name_____ :Roll No_____ :Section_____

Experiment No. 06

Lab 06 –String Revisited, Formatted Output and Getting and Formatting the Date and Time

Lab Objectives:

1. Strings, Revisited
2. Formatted Output
3. Getting and Formatting the Date and Time

1. String, Revisited

1.1 Background

we introduced the string class str. Our goal then was to show that Python supported values other than numbers. We showed how string operators make it possible to write string expressions and process strings in a way that is as familiar as writing algebraic expressions. We also used strings to introduce the indexing operator []. In this section we cover strings and what can be done with them in more depth.We show, in particular, a more general version of the indexing operator and many of .the commonly used string methods that make Python a strong text-processing tool

1.2 String Representations

We already know that a string value is represented as a sequence of characters that is enclosed :within quotes, whether single or double quotes

"!Hello, World" >>>
'!Hello, World'
'hello' >>>
'hello'

1.3 The Indexing Operator, Revisited

We have already introduced the indexing operator []:

```
>>> s = 'hello'
>>> s[0]
'h'
```

The indexing operator takes an index i and returns the single-character string consisting of the character at index i. The indexing operator can also be used to obtain a *slice* of a string. For example:

```
>>> s[0:2]
'he'
```

The expression s[0:2] evaluates to the slice of string s *starting* at index 0 and ending *before* index 2. In general, s[i:j] is the substring of string s that starts at index i and ends at index j-1. Here are more examples.

>>> s[3:4]

'l :Student Name_____ :Roll No_____ :Section_____

```
>>> s[-3:-1]
'll'
```
The last example shows how to get a slice using negative indexes: The substring obtained starts at index -3 and ends *before* index -1 (i.e., at index -2). If the slice we want starts at the first character of a string, we *can* drop the first index:

```
>>> s[:2]
'he'
```
In order to obtain a slice that ends at the last character of a string, we must drop the second index:
```
>>> s[-3:]
'llo'
```

String Methods 1.4

The string class supports a large number of methods. These methods provide the developer with a text-processing toolkit that simplifies the development of text-processing applications. Here we cover .some of the more commonly used methods

We start with the string method find(). When it is invoked on string s with one string input argument target, it checks whether target is a substring of s. If so, it returns the .index (of the first character) of the first occurrence of string target; otherwise, it returns -1 For example, here is how method find() is invoked on string message using target string :'top secret'

```
message = '"This message is top secret and should not >>>
'"be divulged to anyone without top secret clearance
message.find('top secret') >>>
16
```

Index 16 is output by method find() since string 'top secret' appears in string message .starting at index 16
The method count(), when called by string s with string input argument target, returns :the number of times target appears as a substring of s. For example

```
message.count('top secret') >>>
2
```
.The value 2 is returned because string 'top secret' appears twice in message
The function replace(), when invoked on string s, takes two string inputs, old and new, and outputs a copy of string s with every occurrence of substring old replaced by :string new. For example
```
message.replace('top', 'no') >>>
This message is no secret and should not\n'
'be divulged to anyone without no secret clearance
```
:Has this changed the string message? Let’s check

:Student Name_____ :Roll No_____ :Section_____

print(message) >>>
This message is top secret and should not be divulged to anyone without top secret clearance So string message was not changed by the replace() method. Instead, a copy of message, with

appropriate substring replacements, got returned. This string cannot be used later on because we have not assigned it a variable name. Typically, the replace() method would be used in an assignment :statement like this

public = message.replace('top', 'no') >>>
print(public) >>>

This message is no secret and should not be divulged to anyone without no secret clearance Recall that strings are immutable (i.e., they cannot be modified). This is why string method replace() returns a (modified) copy of the string invoking the method rather than changing the string. In the next :example, we showcase a few other methods that return a modified copy of the string

>>> 'message = 'top secret
()message.capitalize >>>
'Top secret'
()message.upper >>>
'TOP SECRET'

Method capitalize(), when called by string s, makes the first character of s uppercase; method upper() makes all the characters uppercase. The very useful string method split() can be called on a string in :order to obtain a list of words in the string

()this is the text'.split' >>>
['this', 'is', 'the', 'text']

In this statement, the method split() uses the blank spaces in the string 'this is the text' to create word substrings that are put into a list and returned. The method split() can also be called with a delimiter string as input: The delimiter string is used in place of Revisited 97 the blank space to break up the string. For example, to break up the string

'x = '2;3;5;7;11;13 >>>

:into a list of number, you would use ';' as the delimiter

(';')x.split >>>
['13' ,'11' ,'7' ,'5' ,'3' ,'2']

Finally, another useful string method is translate(). It is used to replace certain characters in a string with others based on a mapping of characters to characters. Such a mapping is constructed using a special type of string method that is called not by a string object but
:by the string class str itself
table = str.maketrans('abcdef', 'uvwxyz') >>>

,The variable table refers to a "mapping" of characters a,b,c,d,e,f to characters u,v,w,x,y,z respectively.For our purposes here, it is enough to understand its use as an argument to the method :()translate

```
fad'.translate(table)' >>>
'zux'
desktop'.translate(table)' >>>
'xysktop'
```

The string returned by translate() is obtained by replacing characters according to the mapping described by table. In the last example, d and e are replaced by x and y, but the .other characters remain the same because mapping table does not include them

A partial list of string methods is shown in Table 6.1. Many more are available, and to :view them all, use the help() tool

```
help(str) >>>
...
```

Usage	Returned Value
`s.capitalize()`	A copy of string `s` with the first character capitalized if it is a letter in the alphabet
`s.count(target)`	The number of occurrences of substring `target` in string `s`
`s.find(target)`	The index of the first occurrence of substring `target` in string `s`
`s.lower()`	A copy of string `s` converted to lowercase
`s.replace(old, new)`	A copy of string `s` in which every occurrence of substring `old`, when string `s` is scanned from left to right, is replaced by substring `new`
`s.translate(table)`	A copy of string `s` in which characters have been replaced using the mapping described by `table`
`s.split(sep)`	A list of substrings of strings `s`, obtained using delimiter string `sep`; the default delimiter is the blank space
`s.strip()`	A copy of string `s` with leading and trailing blank spaces removed
`s.upper()`	A copy of string `s` converted to uppercase

:Student Name_____ :Roll No_____ :Section_____

4.2 Formatted Output

The results of running a program are typically shown on the screen or written to a file. Either way, the results should be presented in a way that is visually effective. The Python output formatting tools help achieve that. In this section we learn how to format output using features of the print() function and the string format() method. We also look at how strings containing a date and time are interpreted and created.

4.2.1 Function print()

The print() function is used to print values onto the screen. Its input is an object and it prints a *string representation* of the object's value.

```
>>> n = 5
>>> print(n)
5
```

Function print() can take an arbitrary number of input objects, not necessarily of the same type. The values of the objects will be printed in the same line, and blank spaces (i.e., characters ' ') will be inserted between them:

```
>>> r = 5/3
>>> print(n, r)
5 1.66666666667
>>> name = 'Ida'
>>> print(n, r, name)
5 1.66666666667 Ida
```

The blank space inserted between the values is just the default separator. If we want to insert semicolons between values instead of blank spaces, we can do that too. The print() function takes an optional separation argument sep, in addition to the objects to be printed:

```
>>> print(n, r, name, sep=';')
5;1.66666666667;Ida
```

The argument sep=';' specifies that semicolons should be inserted to separate the printed values of n, r, and name. In general, when the argument sep=<some string> is added to the arguments of the print() function, If we want to print the values in separate lines, the separator should be the new line character,
'\n':

```
>>> print(n, r, name, sep='\n')
5
1.66666666667
Ida
```

:Student Name :Roll No :Section

The print() function supports another formatting argument, end, in addition to sep. Normally, each successive print() function call will print in a separate line:

```
>>> for name in ['Joe', 'Sam', 'Tim', 'Ann']:
print(name)
Joe
Sam
Tim
Ann
```

The reason for this behavior is that, by default, the print() statement appends a new line character (\n) to the arguments to be printed. Suppose that the output we really want is:

Joe! Sam! Tim! Ann!

When the argument end=<some string> is added to the arguments to be printed, the string <some string> is printed after all the arguments have been printed. If the argument end=<some string> is missing, then the default string '\n', the new line character, is printed instead; this causes the current line to end. So, to get the screen output in the format we want, we need to add the argument end = '! ' to our print() function call:

```
>>> for name in ['Joe', 'Sam', 'Tim', 'Ann']:
print(name, end='! ')
Joe! Sam! Tim! Ann!
```

4.2.2 String Method format()

The sep argument can be added to the arguments of a print() function call to insert the same string between the values printed. Inserting the same separator string is not always what we want. Consider the problem of printing the day and time in the way we expect to see time, given these variables:

```
>>> weekday = 'Wednesday'
>>> month = 'November'
>>> day = 10
>>> year = 2019
>>> hour = 11
>>> minute = 45
>>> second = 33
```

What we want is to call the print() function with the preceding variables as input arguments and obtain something like:

Wednesday, November 10, 2010 at 11:45:33
It is clear that we cannot use a separator argument to obtain such an output. One way to achieve this output would be to use string concatenation to construct a string in the right format:

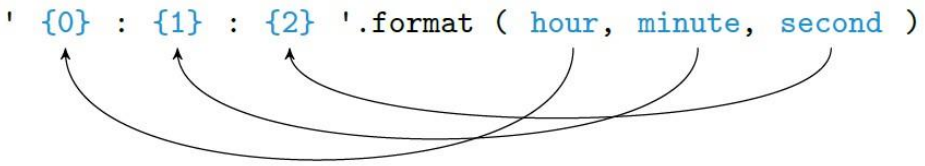
_ :Student Name___ -Roll No- -Section_____

```
>>> print(weekday+', '+month+' '+str(day)+', '+str(year)
    +' at '+str(hour)+':'+str(minute)+':'str(second))
SyntaxError: invalid syntax (<pyshell#36>, line 1)
```

Ooops, There is a mistake, a + before str(second). That fixes it (check it!) but we should not be satisfied. The reason why I messed up is that the approach I used is very tedious and error prone. There is an easier, and far more flexible, way to format the output. The string (str) class provides a powerful class method, format(), for example, in which we only want to print the time:

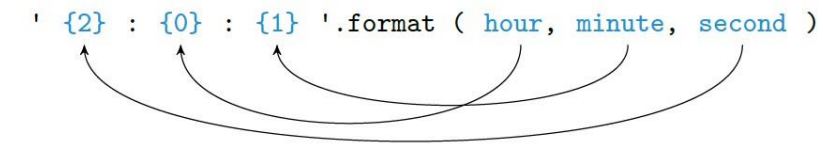
```
>>> '{0}:{1}:{2}'.format(hour, minute, second)
'11:45:33'
```

The objects to be printed (hour, minute, and second) are arguments of the format() method. The string invoking the format() function—that is, the string '{0}:{1}:{2}'— is the format string: It describes the output format. All the characters outside the curly braces—that is, the two colons (':')—are going to be printed as is. The curly braces {0}, {1}, and {2} are *placeholders* where the objects will be printed. The numbers 0, 1, and 2



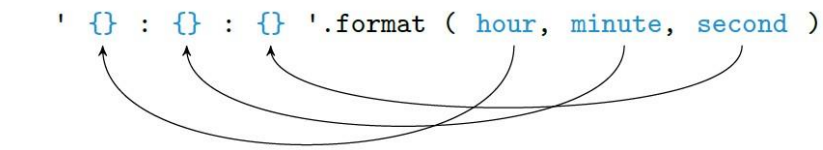
Following Figure shows what happens when we move the indexes 0, 1, and 2 in the previous example:

```
>>> '{2}:{0}:{1}'.format(hour, minute, second)
'33:11:45'
```



The default, when no explicit number is given inside the curly braces, is to assign the first placeholder (from left to right) to the first argument of the format() function, the second placeholder to the second argument, and so on, as shown in following Figure.

```
>>> '{}:{}:{}'.format(hour, minute, second)
'11:45:33'
```



:Student Name _____ :Roll No _____ :Section _____

Let's go back to our original goal of printing the date and time. The format string we need is '{}, {} {}, {} at {}:{}:{}' assuming that the format() function is called on variables weekday, month, day, year, hours, minutes, seconds in that order. We check this (see also following Figure for the illustration of the mapping of variables to placeholders):

```
>>> print('{}, {} {}, {} at {}:{}:{}'.format(weekday, month,
          day, year, hour, minute, second))
Wednesday, March 10, 2010 at 11:45:33
```

4.2.3 Lining Up Data in Columns

To illustrate the issues, let's consider the problem of properly lining up values of functions i^2, i^3 and 2^i for i = 1; 2; 3; . . . Lining up the values properly is useful because it illustrates the very different growth rates of these functions:

i	i**2	i**3	2**i
1	1	1	2
2	4	8	4
3	9	27	8
4	16	64	16
5	25	125	32
6	36	216	64
7	49	343	128
8	64	512	256
9	81	729	512
10	100	1000	1024
11	121	1331	2048
12	144	1728	4096

In this example, we are printing integer values 12 and 354. The format string has a placeholder for with '0:3' inside the braces. The 0 refers to the first argument of the format() function (12), as 12we've seen before. Everything after the ':' specifies the formatting of the value. In this case, 3 indicates that the width of the placeholder should be 3. Since 12 is a two-digit number, an extra blank space is added in front. The placeholder for 354 contains '1:5', so an extra two blank spaces are added in front. When the field width is larger than the number of digits, the default is to right-justify— that is, push the number value to the right. Strings are left-justified. In the next example, a field of width 10 characters is reserved for each argument first and last. Note that extra blanks are added after the string :value

:Student Name_____ :Roll No_____ :Section____

'first = 'Bill >>>
'last = 'Gates >>>
Use format string to print.
' Bill Gates'

The precision is a decimal number that specifies how many digits should be displayed before and after the decimal point of a floating-point value. It follows the field width and a period separates them. In the next example, the field width is 8 but only four digits of the floating-point value are displayed. Where n = 1000 and d = 3.

```
print("Without Precision:", n/d)
print("With Precision:{:8.2}".format(n/d))
print("With Precision:{:8.3}".format(n/d))
print("With Precision:{:8.4}".format(n/d))
print("With Precision:{:8.5}".format(n/d))
print("With Precision:{:8.6}".format(n/d))
```

The type determines how the value should be presented. The available integer presentation types are listed in Table 4.2. We illustrate the different integer type options on integer value 10.

```
print("With character b:{:b}".format(n))
print("With character c:{:c}".format(n))
print("With character d:{:d}".format(n))
print("With character o:{:o}".format(n))
print("With character x:{:x}".format(n))
print("With character X:{:X}".format(n))
```

Two of the presentation-type options for floating-point values are f and e. The type option f displays the value as a fixed-point number (i.e., with a decimal point and fractional part).

```
print("Without Precision:", n/d)
print("With Precision 2f:{:8.2f}".format(n/d))
print("With Precision 3f:{:8.3f}".format(n/d))
print("With Precision 4f:{:8.4f}".format(n/d))
print("With Precision 5f:{:8.5f}".format(n/d))
print("With Precision 6f:{:8.6f}".format(n/d))
print("With Precision 4e:{:8.4e}".format(n/d))
```

Type	Explanation
b	Outputs the number in binary
c	Outputs the Unicode character corresponding to the integer value
d	Outputs the number in decimal notation (default)
o	Outputs the number in base 8
x	Outputs the number in base 16, using lowercase letters for the digits above 9
X	Outputs the number in base 16, using uppercase letters for the digits above 9

:Student Name_____ :Roll No_____ :Section_____

(3 / 5)format.'{6.2f:{' >>>
'1.67 '
In this example, the format specification ':6.2f' reserves a minimum width of 6 with exactly two digits past the decimal point for a floating-point value represented as a fixedpoint number. The type option :e represents the value in scientific notation in which the exponent is shown after the character e

(3 / 5)}:e{'.format' >>>
'1.666667e+00'
.This represents 1:666667 . 10^0

Now let's go back to our original problem of presenting the values of functions i2, i3, and 2i for i = up to at most 12.We specify a minimum width of 3 for the values . . . ;3 ;2 ;1
.i and 6 for the values of i2, i3, and 2i to obtain the output in the desired format

```
def growthrates(n):
    'prints values of below 3 functions for i = 1, ..., n'
    print(' i   i**2   i**3   2**i')
    formatStr = '{0:2d} {1:6d} {2:6d} {3:6d}'
    for i in range(2, n+1):
        print(formatStr.format(i, i**2, i**3, 2**i))
```

Implement function roster() that takes a list containing student information and prints out a roster, as shown below. The student information, consisting of the student's last name, first name, class, and average course grade, will be stored in that order in a list. Therefore, the input list is a list of lists. Make sure the roster printed out has 10 slots for every string value and 8 for the grade, including 2 .slots for the decimal part

```
>>> students = []
>>> students.append(['DeMoines', 'Jim', 'Sophomore', 3.45])
>>> students.append(['Pierre', 'Sophie', 'Sophomore', 4.0])
>>> students.append(['Columbus', 'Maria', 'Senior', 2.5])
>>> students.append(['Phoenix', 'River', 'Junior', 2.45])
>>> students.append(['Olympis', 'Edgar', 'Junior', 3.99])
>>> roster(students)
Last       First     Class      Average Grade
DeMoines   Jim       Sophomore     3.45
Pierre     Sophie    Sophomore     4.00
Columbus   Maria     Senior        2.50
Phoenix    River     Junior        2.45
Olympia    Edgar     Junior        3.99
```

Getting and Formatting the Date and Time 4.3

Getting and Formatting the Date and Time Programs often need to interpret or produce strings that contain a date and time. In addition, they may also need to obtain the current time. The current date

:Student Name :Roll No :Section

and time are obtained by "asking" the underlying operating system. In Python, the time module provides an API to the operating system time utilities as well as tools to format date and time values. To see how to use it, we start by importing the time module:

```
>>> import time
```

Several functions in the time module return some version of the current time. The time() function returns the time in seconds since the epoch:

```
>>> time.time()
1268762993.335
```

You can check the epoch for your computer system using another function that returns the time in a format very different from time():

```
>>> time.gmtime(0)
```

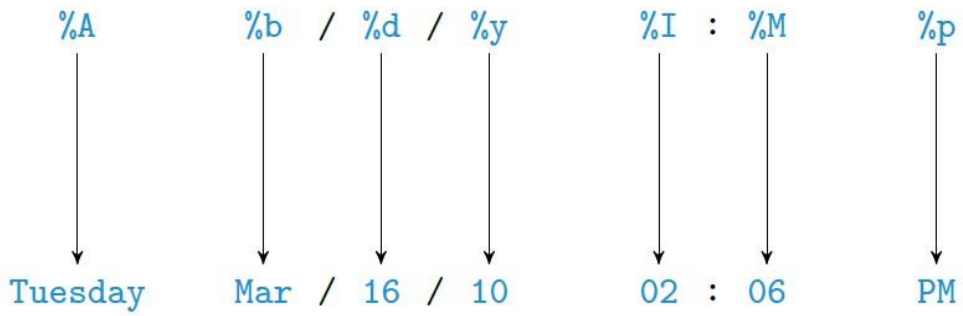
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)

```
>>> time.localtime()
```

time.struct_time(tm_year=2010, tm_mon=3, tm_mday=16, tm_hour= 13, tm_min=50, tm_sec=46, tm_wday=1, tm_yday=75, tm_isdst=1)

The output format is not very readable (and is not designed to be). Module time provides a formatting function strftime() that outputs time in the desired format. This function takes a format string and the time returned by gmtime() or localtime() and outputs the time in a format described by the format string. Here is an example, illustrated in Figure.

```
>>> time.strftime('%A %b/%d/%y %I:%M %p', time.localtime())
'Tuesday Mar/16/10 02:06 PM'
```



:Student Name_____ :Roll No_____ :Section_____

In this example, strftime() prints the time returned by time.localtime() in the format specified by the format string '%A %b/%d/%y %I:%M %p'. The format string includes *directives* %A, %b, %d, %y, %I, %M, and %p that specify what date and time values to output at the directive's location, using the mapping shown in Table 4.3. All the other characters (/, :, and the blank spaces) of the format string are copied to the output as is.

Directive	Output
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%d	The day of the month as a decimal number between 01 and 31
%H	The hours as a number between 00 and 23
%I	The hours as a number between 01 and 12
%M	The minutes as a number between 00 and 59
%p	AM or PM
%S	Seconds as a number between 00 and 61
%y	Year without century as a number between 00 and 99
%Y	Year as a decimal number
%Z	Time zone name

Student Name:_____ Roll No:_____ Section:_____

Programming Exercise

1. Construct the strings by using the string time format function strftime ()

a) (‘Monday, November 25 2019')
b) ('09:40 PM Central Daylight Time on 11/25/2019')
c) ('I will meet you on Thu July 13 at 09:40 PM.')

2. Assuming that variable forecast has been assigned string 'It will be a sunny day today'
Write Python function corresponding to these assignments:

(a) To variable count, the number of occurrences of string 'day' in string forecast.
(b) To variable weather, the index where substring 'sunny' starts.
(c) To variable change, a copy of forecast in which every occurrence of substring 'sunny' is replaced by 'cloudy'.

3. Write function even() that takes a positive integer n as input and prints on the screen all numbers between, and including, 2 and n divisible by 2 or by 3, using this output format:

```
>>> even(17)
2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16,
```

4. Assume variables first, last, street, number, city, state, zip code have already been assigned in a function mailaddress(). When you pass all the parameters it generates a mailing label:

Syed Faisal Ali
Asst. Prof.
Department of Computer Science
Usman Institute of Technology
Gulshan e Iqbal, 75300
Karachi.

5. Write a function month() that takes a number between 1 and 12 as input and returns the three-character abbreviation of the corresponding month. Do this without using an if statement, just string operations. Hint: Use a string to store the abbreviations in order.

```
>>> month(1)
'Jan'
>>> month(11)
'Nov'
```

6. Implement function cheer() that takes as input a team name (as a string) and prints a cheer as shown:

```
>>> cheer('uitians')
How do you spell winner?
I know, I know!
U I T I A N S !
And that's how you spell winner!
Go Uitians!
```

Student Name:_____ Roll No:_____ Section:_____

7. Design an application in of small POS (Point of Sale) in which you generate burger type, quantity, rate and amount line by line at the end it will print the total sum and 13 % GST. Keep in mind that each receipt has username, and date of print. Create a function which can solve your problem.

8. Create a function which will take a start point and end point from user and convert the numbers in binary, octal, decimal, and hexa-decimal with proper formatting style.
The first line will be headings and the rest of the line with proper space between each type.

9. Create a dictionary with name, roll number, semester, semester fees, miscellaneous fees, sports fees per semester etc and it will write paid amount with total. This is for one student; you need to generate a proper receipt for at least 3 students but same formatting just like fees vouchers generated from school. Student copy, University Copy and Bank Copy.

10. Compare two brands of petrol prices in Pakistan:
PSO and Shell
Pakistan Oil Prices, Pakistan Petroleum Prices and current Petrol Prices
PSO

Type	Price
Premium (Super)	Rs. 114.24 /Ltr
High Speed Diesel	Rs. 127.41 /Ltr
Light Speed Diesel	Rs. 85.33 /Ltr
Kerosene Oil	Rs. 97.18 /Ltr

Shell

Product Name	Rs./litre
E10 Gasoline	111.74
Altron Premium	114.24
Action + Diesel	127.41
LDO	85.33
SKO	97.18

Find that if a user can extract the amount of Petrol for a month where monthly petrol is 500 liters, 300 liters' diesel and 200 liters of kerosene oil. What is the difference between the amount in both? Keep in mind proper formatting is required.