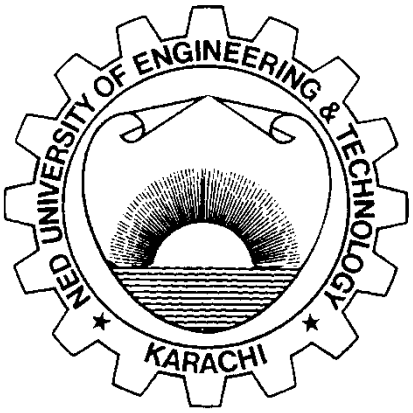
Practical Workbook
CS-115
Computer Programming



Name _____

Year _____

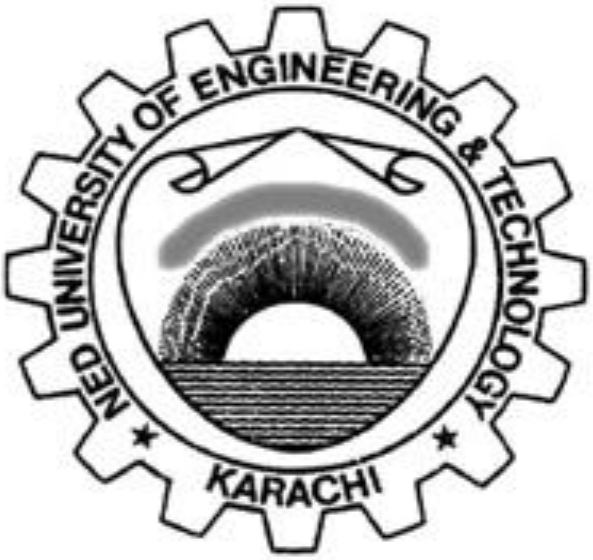
Batch _____

Roll No _____

Department: _____

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

Practical Workbook
CS-115
Computer Programming



Prepared by:

Mr. Kashif Asrar
Ms. Ibshar Ishrat

Revised in:

September 2019

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

Introduction

This workbook for CS-115 Computer Programming introduces basic as well as intermediate level concepts of programming using Python language. Each lab session begins with a brief theory of the topic. Many details have not been incorporated as the same is to be covered in Theory classes. The Exercise section follows this section.

The Course Profile of CS-115 Computer Programming lays down the following Course Learning Outcome:

"Practice computer programming using constructs of a high level language. (C3, PLO-5)"

All lab sessions of this workbook have been designed to assist the achievement of the above CLO. A rubric to evaluate student's performance has been provided at the end of the workbook.

The Workbook has been arranged as fourteen labs starting with a practical on the Introduction to programming environment and fundamentals of programming language. Next few lab sessions deal with familiarization with different data types and operations supported by those data types. Single stepping; an efficient debugging and error detection technique is discussed in Lab session 4. Next lab session covers decision making in programming and its application. Lab session 6 and 7 introduce the concepts of loops with different examples to use them in programming.

Lab session 8 introduces a new tool 'PyCharm' for execution of python projects and scripts. Function declaration and definition concepts and examples are discussed in lab sessions 9, 10 and 11. The next lab session deals with the advanced data type in python named 'tuples' for which Project Jupyter- (a web based application to code scripts and run projects) would be used.

In the final lab operations on files like reading and writing have been discussed. These operations enable the users to handle not only large amount of data, but also data of different types (integers, characters etc.) and to do so efficiently.

Contents

Lab Session 01

Explore programming fundamentals and Python IDLE

COMPUTER PROGRAMMING

Computer programming is the act of writing computer programs, which are a sequence of instructions written using a computer programming language to perform a specified task by the computer. There exists a large number of high-level languages. Some of these include BASIC, C, C++, FORTRAN, Java, Pascal, Perl, PHP, Python, Ruby, and Visual Basic etc.

INTRODUCTION TO PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, makes it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

PROGRAM DEVELOPMENT WITH PYTHON IDE

This lab session introduces the Integrated Development Environment (IDE) of Python and shows how to enter, edit, save, retrieve, compile, link, and run a python program in such an environment.

Hello-World Program (Approach-1)

After installation of Python, follow following steps to develop and execute python program

- Create a python script file by replacing the extension of text file (.txt) with (.py).
- Right click on the file (say first.py) and select “Edit with IDLE”.
- The file will be opened as shown in figure 1.1

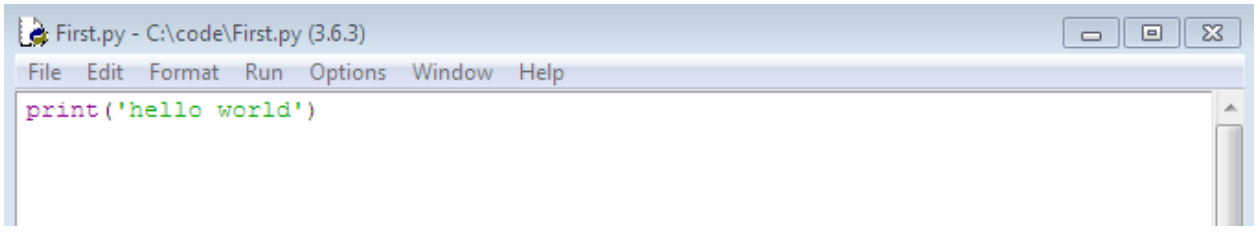


Fig. 1.1

- Type the program and click on “run > run module “or press F5 to execute the program. The window (on the next page) will appear showing the output of program

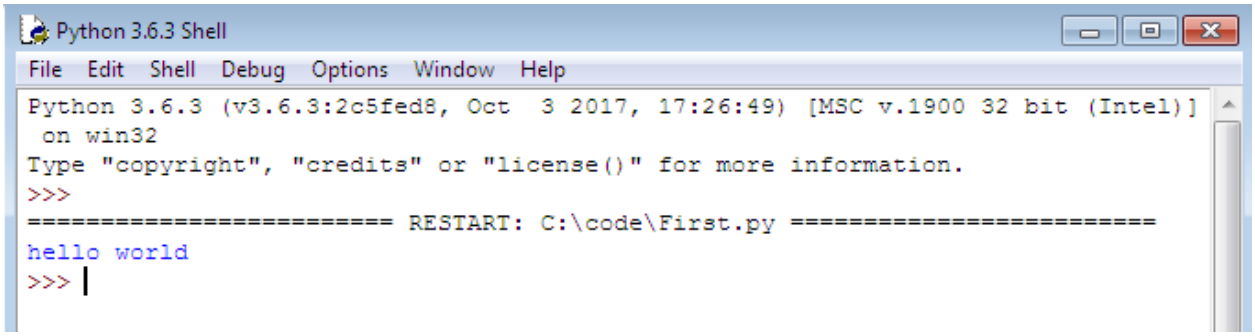


Fig. 1.2

- After the prompt (>>>), any command typed will be executed as soon as Enter key will be pressed.

Hello-World Program (Approach-2)

After installation of Python, perform the following steps to develop and execute the python program.

- Create a python script file by replacing the extension of text file (.txt) with (.py).
- Open command prompt by clicking on command prompt from the start> All Programs > Accessories

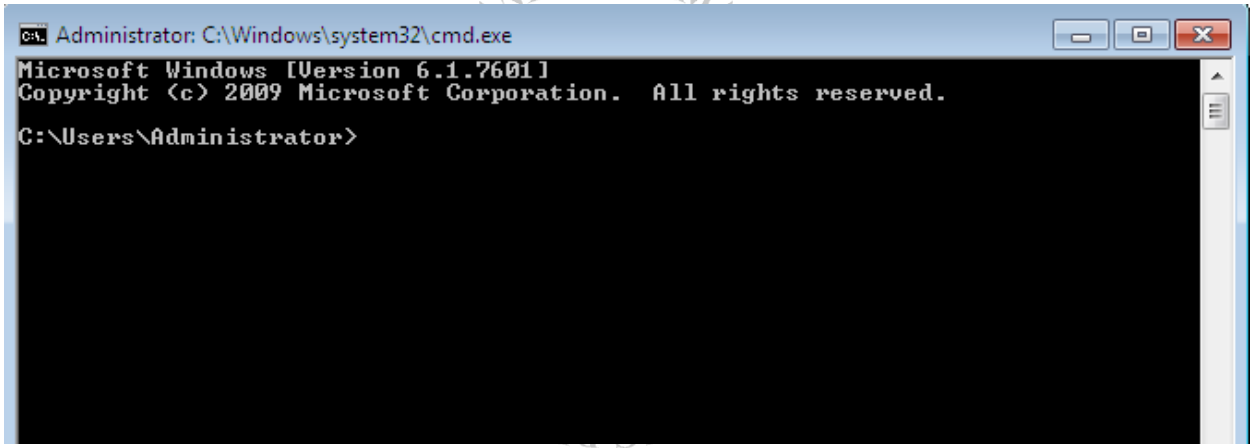


Fig. 1.3

- Change the path of DOS to the folder containing First.py with 'cd' command

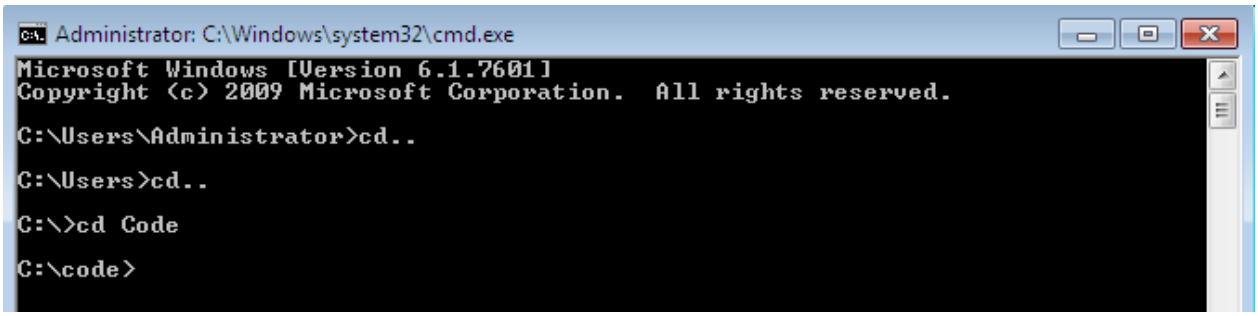


Fig. 1.4

- Run the script by using command 'python First.py'

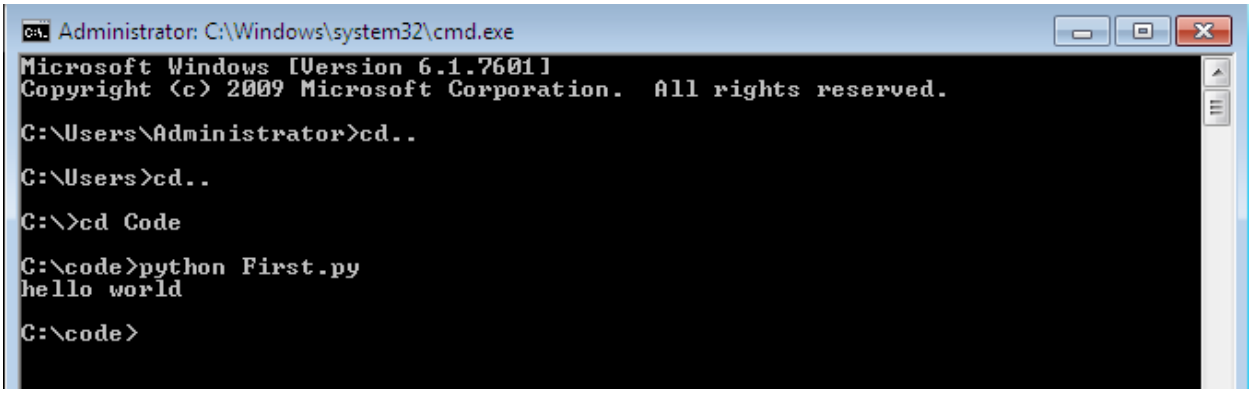


Fig. 1.5

PYTHON IDLE OPTIONS

Format Menu

1. Indent Region

   Shifts selected lines right by the indent width (default 4 spaces).

2. De-indent Region

   Shifts selected lines left by the indent width (default 4 spaces).

3. Comment Out Region

   Inserts ## in front of selected lines.

4. Uncomment Region

   Removes leading # or ## from selected lines.

5. Tabify Region

   Turns *leading* stretches of spaces into tabs. (Note: We recommend using 4 space blocks to indent Python code.)

6. Untabify Region

   Turns *all* tabs into the correct number of spaces.

7. Toggle Tabs

   Opens a dialog to switch between indenting with spaces and tabs.

8. New Indent Width

   Opens a dialog to change indent width. The accepted default by the Python community is 4 spaces.

9. Format Paragraph

   Reformats the current blank-line-delimited paragraph in comment block or multiline string or selected line in a string. All lines in the paragraph will be formatted to less than N columns, where N defaults to 72.

10. Strip trailing whitespace

    Removes any space characters after the last non-space character of a line.

RUN Menu

1. Python Shell

   Open or wake up the Python Shell window.

2. Check Module

   Check the syntax of the module currently open in the Editor window. If the module has not been saved, IDLE will either prompt the user to save or auto-save, as selected in the General tab of the Idle Settings dialog. If there is a syntax error, the approximate location is indicated in the Editor window.

3. Run Module

   Do Check Module (above). If no error, restart the shell to clean the environment then execute the module. Output is displayed in the Shell window. Note that output requires use of print or write. When execution is complete, the Shell retains focus and displays a prompt. At this point, one may interactively explore the result of execution. This is similar to executing a file with python –i file at a command line.

Options Menu

1. Configure IDLE
   Open a configuration dialog and change preferences for the following:
   Fonts, indentation, key bindings, text color themes, startup windows and size, additional help sources, and extensions. To use a new built-in color theme (IDLE Dark) with older IDLEs, save it as a new custom theme.
   Non-default user settings are saved in a .idlerc directory in the user's home directory. Problems caused by bad user configuration files are solved by editing or deleting one or more of the files in .idlerc.

2. Code Context (toggle)(Editor Window only)

   Open a pane at the top of the edit window which shows the block context of the code which has scrolled above the top of the window.

EXERCISE

1. Explore the following Python folders and run random scripts from the folders. Observe the output. Attach screenshot of any 3 outputs.
   i) Lib
   ii) Tools -> demo

2. Change the IDLE theme, font-size, font-color and highlights. Attach screenshot of your configuration.

Attach screenshot of outputs here

Lab Session 02

Practice operations on integer and string data types

OPERATIONS ON IDLE

Executing Mathematical Operators on IDLE

Routine mathematical operations like subtraction, multiplication and division can be performed in the similar way as addition operation performed below:

```
>>> 123+456  #Addition
579
>>> 123**2  #Power
15129
>>> 2.0 >= 1 # Greater than or equal: mixed-type 1 converted to 1.0
True
>>> 2.0 == 2.0 # Equal value
True
>>> 2.0 != 2.0 # Not equal value
False
```

Executing String Operators on IDLE

```
>>> s = 'a\nb\tc'
>>> s
'a\nb\tc'
>>> print(s)
a
b c
>>> S = 'Spam' # Make a 4-character string, and assign it to a name
>>> len(S) # Length
4
>>> S[0] # The first item in S, indexing by zero-based position
'S'
```

In Python, we can also index backward, from the end—positive indexes count from the left, and negative indexes count back from the right:

```
>>> S[-1] # The last item from the end in S
'm'
>>> S[-2] # The second-to-last item from the end
>>> S # A 4-character string
'Spam'
```

Data Type Conversion

```
>>> "42" + 1
TypeError: Can't convert 'int' object to str implicitly
>>> int("42"), str(42) # Convert from/to string
(42, '42')
>>> S = "42"
>>> I = 1
>>> S + I
TypeError: Can't convert 'int' object to str implicitly
>>> int(S) + I # Force addition
43
```

```
>>> S + str(I) # Force concatenation
'421'
```

Slicing

```
>>> S[1:3] # Slice of S from offsets 1 through 2 (not 3)
'pa'
>>> S[1:] # Everything past the first (1:len(S))
'pam'
>>> S # S itself hasn't changed
'Spam'
```

Strings are *immutable* in Python i.e. they cannot be changed in place after they are created. For example, a string can't be changed by assigning to one of its positions, but new string can always be assigned to the same string. Because Python cleans up old objects

```
>>> S
'Spam'
>>> S[0] = 'z' # Immutable objects cannot be changed
...error text omitted...
TypeError: 'str' object does not support item assignment
>>> S = 'z' + S[1:] # But we can run expressions to make new objects
>>> S
'zpam'
>>> 'abc' + 'def' # Concatenation: a new string
'abcdef'
>>> 'Ni!' * 4 # Repetition: like "Ni!" + "Ni!" + ...
'Ni!Ni!Ni!Ni!'
```

Extended Slicing

The third parameter in square bracket defines

- Difference between the indexes to be printed on output
- Direction of access i.e. negative difference define the access direction from right to left

```
s='Computer'
a=s[::-1]
print(a)
#Output:retupmoC
a=s[1:5:1]
print(a)
# Output:ompu
a=s[1:5:2]
print(a)
# Output:op
a=s[5:1:-1]
print(a)
# Output:tupm
```

Input Function

input()

The input function reads a line from provided in parenthesis and converts it to a string (stripping a trailing newline), and returns that to the output screen.

EXCERCISE

1. Implement quadratic equation to find out both values. Provide at least three set of values for a, b and c to get the output. A, b and c will be provided by user as input (Hint: use int function)

Write the program here:

a= , b= ,c= X1= X2=

a= , b= ,c= X1= X2=

a= , b= ,c= X1= X2=

2. Write down the slicing statements to generate following outputs when string='COMPUTERPROGRAMMING':

Output: PUTER Statement:

Output: GRAMM Statement:

Output: PROGRAM Statement:

Output: COMPUTER Statement:

3. Write down the extended slicing statements to generate following outputs when string='COMPUTERPROGRAMMING'

Output: RETUP Statement:

Output: MMARG Statement:

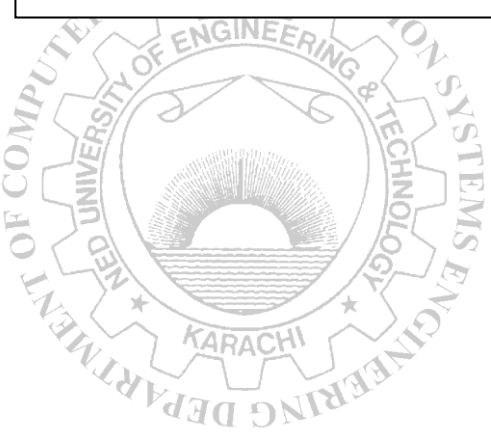
Output: MARGORP Statement:

Output: RETUPMOC Statement:

4. Develop the script to print the following pattern when string is ‘COMPUTER’

```
COMPUTERS
OMPUTERS
MPUTERS
PUTERS
UTERS
TERS
ERS
RS
S
```

Write the program here:



Lab Session 03

Use Decision making in programming. (if –else & conditional operator)

DECISION MAKING STRUCTURES

Normally, the program flows along line by line in the order in which it appears in source code. But, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. you have to make decision in the program.

General Format

```
if test1:                    # if test
      statements1                  # Associated block
elif test2:                        # Optional elifs
      statements2
else:                              # Optional else
      statements3
```

The indentation (blank whitespace all the way to the left of the two nested statements here) is the factor that defines which code block lies within the condition statement. Python doesn't care how indents can be inserted (either spaces or tabs may be used), or how much a statement can be indented (any number of spaces or tabs can be used). In fact, the indentation of one nested block can be totally different from that of another. The syntax rule is only that for a given single nested block, all of its statements must be indented the same distance to the right. If this is not the case, a syntax error will appear, and code will not run until its indentation is repaired to be consistent. Python almost forces programmers to produce uniform, regular, and readable code

The one new syntax component in Python is the colon character (:). All Python *compound statements* that have other statements nested inside them—follow the same general pattern of a header line terminated in a colon, followed by a nested block of code usually indented underneath the header line

EXCERCISE

1. Write a program that takes a positive integer as input from user and checks whether the number is even or odd, and displays an appropriate message on the screen. [**Note:** For negative numbers, program does nothing.]

2. Write a script to print the grade (according to given table) when user enters his/her marks.

Grade	Grade Point	Marks
A	4.0	88 – 100
A –	3.7	80 – 87
B +	3.4	75 – 79
B	3.0	70 – 74
B –	2.7	67 – 69
C +	2.4	64 – 66
C	2.0	60 – 63
C –	1.7	57 – 59
D +	1.4	54 – 56
D	1.0	50 – 53

Attach code and output here:

3. Write a program that displays "Kamran Akmal" on output, if score >30, Shoaib Akhtar, if 20<score <30, and Shahid Afridi if 10<score <20.

Write the program here and attach output:

4. Write a program that takes password from user as input. Validate the password on the following criteria.
   'Password length between 7 to 15 characters which contain at least one numeric digit and a special character is acceptable.'

Write the program here and attach output:

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Lab Session 04

Carry out Debugging of Programs through IDLE

TYPES OF ERRORS

There are generally two types of errors namely syntax and logical errors. Syntax errors occur when a program does not conform to the grammar of a programming language, and the compiler cannot compile the source file. Logical errors occur when a program does not do what the programmer expects it to do. Syntax errors are usually easy to fix because the compiler can detect these errors. The logical errors might only be noticed during runtime. Because logical errors are often hidden in the source code, they are typically harder to find than syntax errors. The process of finding out defects (logical errors) in the program and fixing them is known as debugging. Debugging is an integral part of the programming process.

Program Debugging With IDLE

1. Open Python shell
2. Go to **file>New** and open a python script file
3. Write a program on that file

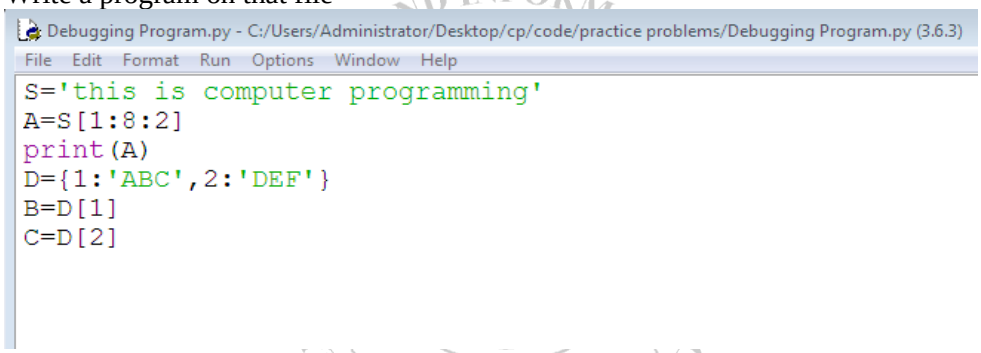


Fig. 4.1

4. Go to Python Shell and select the **Debug** option. A window will appear as shown below

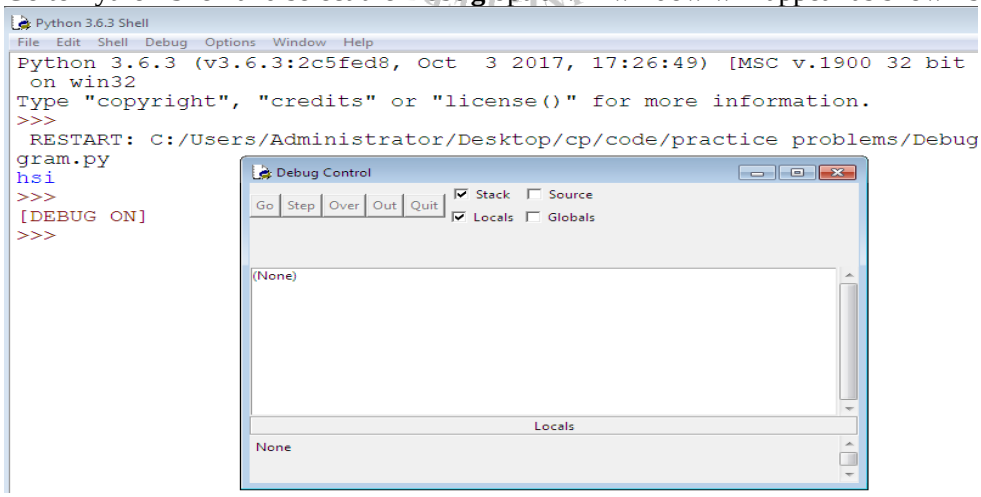


Fig. 4.2

5. Set break point by right clicking on the particular line.

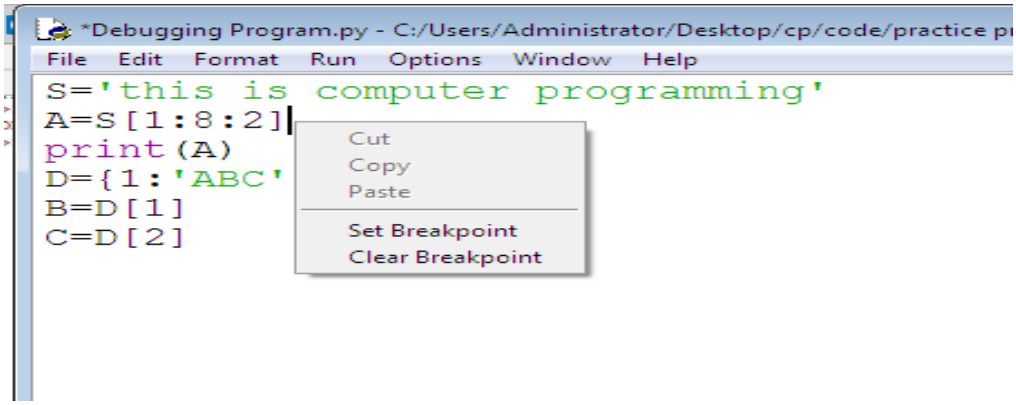


Fig 4.3

6. Now go to python script and click **Run** and notice the line highlighted by blue. Note that the Debug Control window is opened and that the blue line states that the line 1 "S='this is computer programming'" is ready to be executed

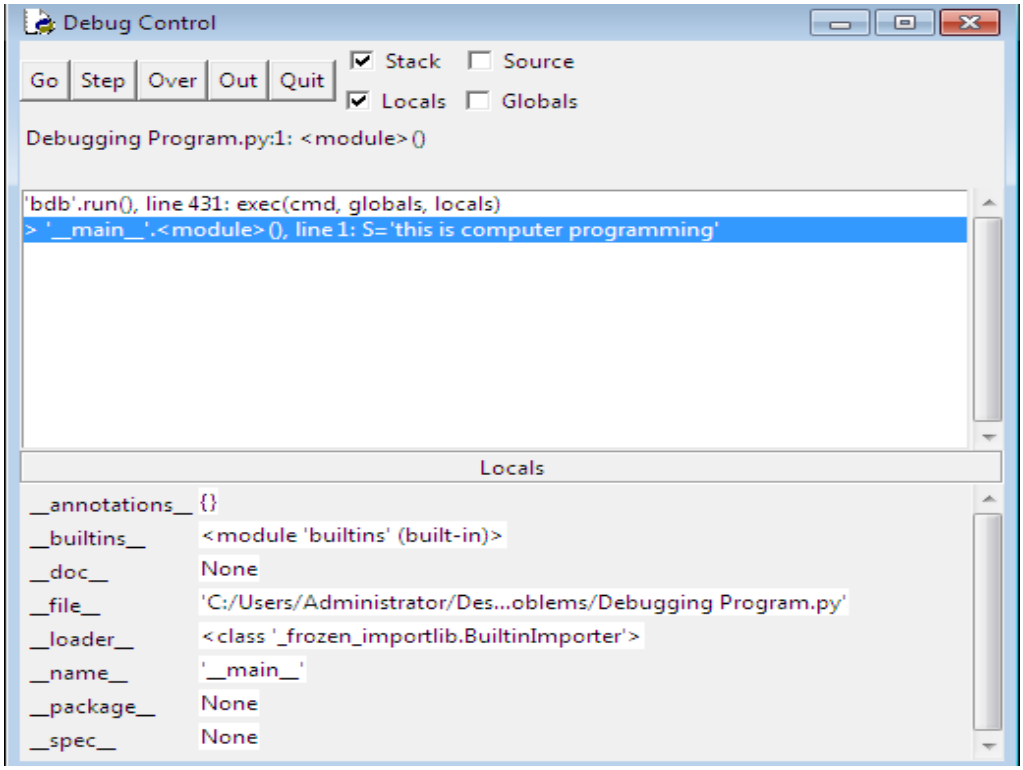


Fig. 4.4

a. **Go** button will make the program run at normal speed until a breakpoint is encountered (or input is requested or the program finishes).

b. **Step** button is used to step through your code, one line at a time.

c. There is a pane "Locals" which shows the value of A and S. This is useful in several ways. It shows the values of variables as they change, and it shows the **types** of variables.

d. **Over** means that if the statement to be executed has a function call in it, go off and do the function call without showing any details of the execution or variables, then return and give the human control again, "step over the function"

e. **Out** assumes you are in some function's code, finish execution of the function at normal speed, return from the function and then give the human control again, "step out of the function"

f. **Quit** stops the execution of the entire program

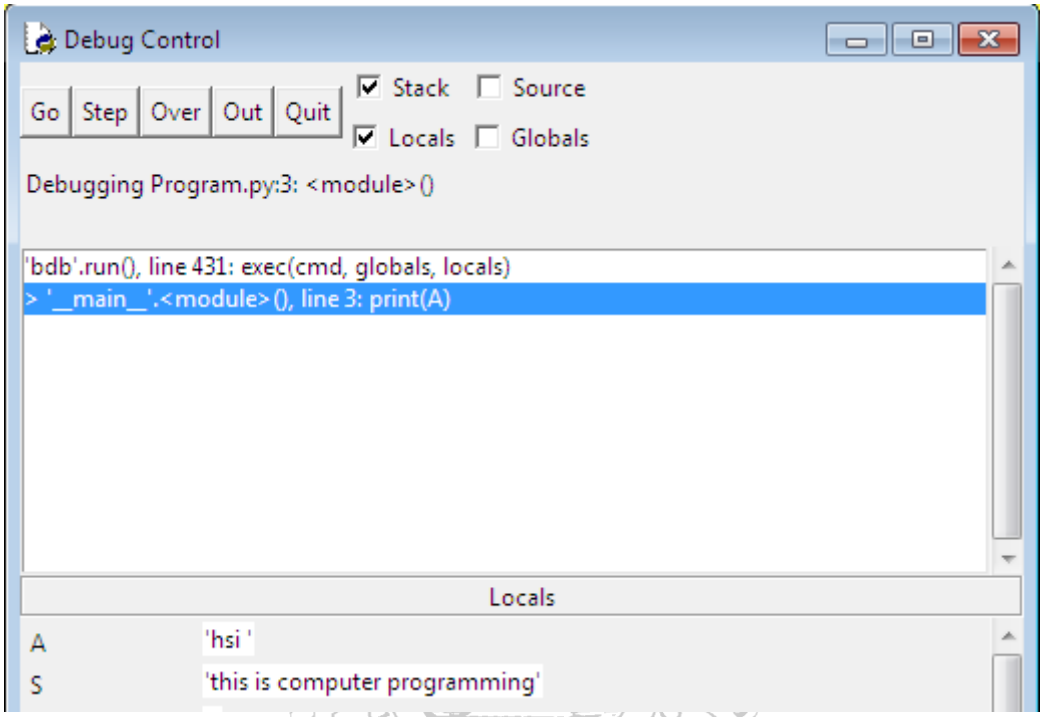


Fig. 4.4

Summary

- Setting breakpoints
- Stepping through the source code one line at a time
- Inspecting the values of variables as they change
- Making corrections to the source as bugs are found
- Rerunning the program to make sure, the fixes are correct

EXCERCISE

1. Write a script that performs at least 5 slicing operation at different position on a string (your Firstname_LastName) saved in a variable. Each slice operation must be saved in a different variable. Debug the program to show the assignment of values to variables through 'debug control' window through single stepping.

Attach printout here:

2. Debug at least two programs from previous lab exercises.

Attach printout here:

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Lab Session 05

Use repetitive structure – while loops, for loops & their nesting

FOR LOOP

The Python for loop begins with a header line that specifies an assignment target (or targets), along with the object you want to step through. The header is followed by a block of (normally indented) statements that you want to repeat:

General Format

```
for target in object:          # Assign object items to target
        statements             # Repeated loop body: use target
else:                          # Optional else part
        statements             # If we didn't hit a 'break'
```

When Python runs a for loop, it assigns the items in the iterable object to the target one by one and executes the loop body for each. The loop body typically uses the assignment target to refer to the current item in the sequence as though it were a cursor stepping through the sequence.

WHILE LOOP

While loop repeatedly executes a block of (normally indented) statements as long as a test at the top keeps evaluating to a true value. It is called a "loop" because control keeps looping back to the start of the statement until the test becomes false. When the test becomes false, control passes to the statement that follows the while block. The net effect is that the loop's body is executed repeatedly while the test at the top is true. If the test is false to begin with, the body never runs and the while statement is skipped

General Format

```
while test:                    # Loop test
      statements               # Loop body
else:                          # Optional else
      statements               #  Run  if  didn't  exit  loop  with
break
```

EXCERCISE

1. Develop a program that takes two strings (string-1, string-2) from user as input and compares string-1 character by character with string-2 to print the common and un-common characters of string-1 with respect to string-2

Write the program here and attach the printout of output

2. Write a program to develop a pattern mentioned below:
   *Note: user input will define the number of lines to be generated with maximum number of **

```
*
**
***
****
*****
******
*******
********
*********
*********
********
*******
******
*****
****
***
**
*
```

Write the program here:

3. Develop a program to find out the largest integer in the list given input by user.

Write the program here and attach the printout of output

4. Develop a program to generate the table (till 10) of integer given as input by user.

Write the program here and attach the printout of output

5. Develop a program that takes an integer (end limit of series) from user and print even numbers within the limit specified by the user.

Write the program here and attach the printout of output

6. Develop a program to perform two simple transactions in a bank as long as user enters „y. to continue.

Sample Output:

Enter your ID: ****	Main Menu ********** 1. Deposit Money 2. Withdraw Amount 3. Login as Different User Select your choice ….	*(after completing the selected transaction)* Do you want to continue? [y/Y] _ *(goes to Main Menu, if y/Y is pressed)*

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Write the program here and attach the printout of output

Lab Session 06

Practice Operations on List Data Type object.

PYTHON LIST

In Python, an object of list data type can be a collection of many data types. Python lists have following basic properties:

- *Ordered collections of arbitrary objects*
  From a functional view, lists are just places to collect other objects Lists also maintain a left-to-right positional ordering among the items contained in them (i.e., they are sequences).
- *Accessed by offset*
  A component object of list can be accessed by its position.
- *Variable-length, heterogeneous, and arbitrarily nestable*
  Unlike strings, lists can grow and shrink in place (their lengths can vary), and they can contain any sort of object, not just one-character strings (they're heterogeneous). Because lists can contain other complex objects, they also support arbitrary nesting.
- *Of the category "mutable sequence"*
  Lists are mutable (i.e., can be changed in place) and can respond to all the sequence operations used with strings, such as indexing, slicing, and concatenation. In fact, sequence operations work the same on lists as they do on strings; the only difference is that sequence operations such as concatenation and slicing return new lists instead of new strings when applied to lists. Because lists are mutable, however, they also support other operations that strings don't, such as deletion and index assignment operations, which change the lists in place.

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[2] # Offsets start at zero
'SPAM!'
>>> L[-2] # Negative: count from the right
'Spam'
>>> L[1:] # Slicing fetches sections
['Spam', 'SPAM!']
>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5] # Replacement/insertion
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7] # Insertion (replace nothing)
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = [] # Deletion (insert nothing)
>>> L
[1, 7, 4, 5, 3]
>>> L = [1]
>>> L[:0] = [2, 3, 4] # Insert all at :0, an empty slice at front
>>> L
[2, 3, 4, 1]
>>> L[len(L):] = [5, 6, 7] # Insert all at len(L):, an empty slice
at end
>>> L
[2, 3, 4, 1, 5, 6, 7]
>>> L.extend([8, 9, 10]) # Insert all at end, named method
>>> L
[2, 3, 4, 1, 5, 6, 7, 8, 9, 10]
```

List Method Calls

```
>>> L = ['THIS', 'IS', 'COMPUTER']
>>> L.append('PROGRAMMING') # Append method call: add item at end
>>> L
['THIS', 'IS', 'COMPUTER', 'PROGRAMMING']
>>> L.sort()
>>> L
['COMPUTER', 'IS', 'PROGRAMMING', 'THIS']
>>> L = [1, 2]
>>> L.extend([3, 4, 5]) # Add many items at end (like in-place +)
>>> L
[1, 2, 3, 4, 5]
>>> L.pop() # Delete and return last item (by default: -1)
5
>>> L
[1, 2, 3, 4]
>>> L.reverse() # In-place reversal method
>>> L
[4, 3, 2, 1]
>>> list(reversed(L)) # Reversal built-in with a result (iterator)
[1, 2, 3, 4]
```

EXCERCISE

1. Write commands to perform operations on list L[5,6,8,9,2,1] to convert it into :

   [1,2,5,6,8,9] Command=

   [1,2,6,8,9] Command=

   [1,2,6,8,9,10] Command=

   [10,9,8,6,2,1] Command=

   [8] Command=

2. Write a program to perform operations on the given list to generate following outputs :
   l=['this','is','simple','computer','programming','using','python']

   Sample Output

```
['this', 'is', 'computer', 'programming']

['this','is','simple']

['this','is', 'programming','using','python']

['programming','using','python']

['is', 'this', 'computer', 'programming']
```

3. From given list:

   gadgets = [“Mobile”, “Laptop”, 100, “Camera”, 310.28, “Speakers”, 27.00, “Television”, 1000, “Laptop Case”, “Camera Lens”]

   a) Create separate lists of strings and numbers.

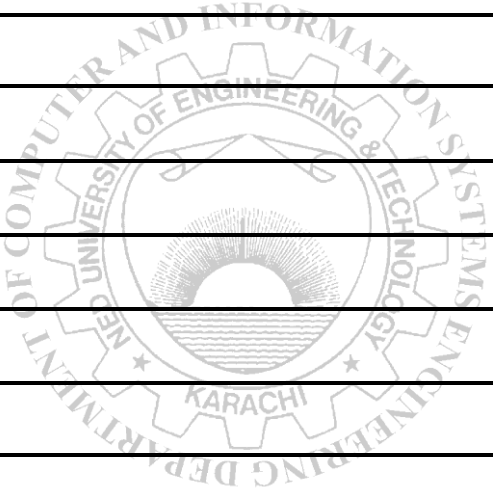
   b) Sort the strings list in ascending order

   c) Sort the strings list in descending order

   d) Sort the number list from lowest to highest

   e) Sort the number list from highest to lowest

4. Produce a code to get first, second best scores from the list L=[86,86,85,85,85,83,23,45,84,1,2,0]



Lab Session 07

Practice Operations on DICTIONARY Data Type object.

PYTHON DICTIONARY

A **dictionary** is an associative array (also known as hashes). Any key of the **dictionary** is associated (or mapped) to a value. The values of a **dictionary** can be any **Python** data type. So **dictionaries** are unordered key-value-pairs with following properties:

- *Accessed by key, not offset position*
  Dictionaries are sometimes called *associative arrays* or *hashes*. They associate a set of values with keys, so an item can be fetched out of a dictionary using the key under which it is originally stored. The same indexing operation can be utilized to get components in a dictionary as in a list, but the index takes the form of a key, not a relative offset.
- *Unordered collections of arbitrary objects*
  Unlike in a list, items stored in a dictionary aren't kept in any particular order. Keys provide the symbolic (not physical) locations of items in a dictionary.
- *Variable-length, heterogeneous, and arbitrarily nestable*
  Like lists, dictionaries can grow and shrink in place (without new copies being made), they can contain objects of any type, and they support nesting to any depth (they can contain lists, other dictionaries, and so on). Each *key* can have just one associated *value*, but that value can be a *collection* of multiple objects if needed, and a given value can be stored under any number of keys.
- *Of the category "mutable mapping"*
  Dictionary allows in place changes by assigning to indexes (they are mutable), but they don't support the sequence operations that work on strings and lists. Because dictionaries are unordered collections, operations that depend on a fixed positional order (e.g., concatenation, slicing) don't make sense. Instead, dictionaries are the only built-in, core type representatives of the *mapping* category— objects that map keys to values. Other mappings in Python are created by imported modules.
- *Tables of object references (hash tables)*
  If lists are arrays of object references that support access by position, dictionaries are unordered tables of object references that support access by key. Internally, dictionaries are implemented as hash tables (data structures that support very fast retrieval), which start small and grow on demand. Moreover, Python employs optimized hashing algorithms to find keys, so retrieval is quick. Like lists, dictionaries store object references (not copies, unless explicitly asked).

Basic Dictionary Operations

```
>>> D = {'this': 2, 'is': 1, 'CP': 3} # Make a dictionary
>>> D['this'] # Fetch a value by key
2
>>> D # Order is "scrambled"
{'this': 2, 'is': 1, 'CP': 3}
>>> len(D) # Number of entries in dictionary
3
>>> 'this' in D # Key membership test alternative
True
>>> list(D.keys()) # Create a new list of D's keys
['this', 'is', 'CP']
```

Changing Dictionaries in Place

```
>>> D
{'this': 2, 'is': 1, 'CP': 3}
>>> del D['this'] # Delete entry
>>> D
{'is': 1, 'CP': 3}
>>> D['Course'] = '4' # Add new entry
>>> D
{'is': 1, 'CP': 3, 'Course': '4'}
>>> D = {'this': 2, 'is': 1, 'CP': 3}
>>> D.values()
dict_values([1, 3, '4'])
>>> D.get('this') # A key that is there
2
>>> print(D.get('game')) # A key that is missing
None
>>> D.get('good', 88)
88
# pop a dictionary by key
>>> D
{'this': 2, 'is': 1, 'CP': 3}
>>> D.pop('CP')
3
>>> D
{'is': 1}
```

Dictionaries as flexible lists:

When a list is used, it is illegal to assign to an offset that is off the end of the list:

```
>>> L = []
>>> L[99] = 'spam'
Traceback (most recent call last):
File "<stdin>", line 1, in ?
IndexError: list assignment index out of range
```

By using integer keys, dictionaries can emulate lists that seem to grow on offset assignment:

```
>>> D = {}
>>> D[99] = 'spam'
>>> D[99]
'spam'
>>> D
{99: 'spam'}
```

EXCERCISE

1. Write a script to develop the given dictionary:

```
D={'CP':'COMPUTER PROGRAMMING',
   'FCE':'FUNDAMENTALS OF COMPUTER ENGINEERING',
   'PST':'PAKISTAN STUDIES',
   'BEE':'BASICS OF ELECTRICAL ENGINEERING',}
```

Write the program here:

2. Write a statement to add 'F.ENG': 'FUNCTIONAL ENGLISH' in the dictionary of Q1.

3. Write a statement to find out whether the given key (Chinese) is the part of dictionary in Q1.

4. Write a statement to print the value by providing the key (‘CP’) to the dictionary in Q1.

5. Write a program that takes a list of multiple choice responses. E.g. [a, b, c] and prints a dictionary of question-response pairs {‘Q1’: ‘a’, ‘Q2’:’b’, ‘Q3’:’c’}.

6. Write a program that will enable a user to enter a restaurant's specials for the day. Each special will have the name of the dish and a price. In a dictionary of menu items, the name will be the key, and the price its value. User should be able to enter a name of a dish and get its price.

Lab Session 08

Explore PyCharm for the execution of python scripts and projects

PYCHARM

Pycharm is an IDE(Integrated Development Environment) developed for the execution of python scripts and projects:
Steps:

1. Go to File > New Project
2. Assign the name to new project (say ‘ My_first’)

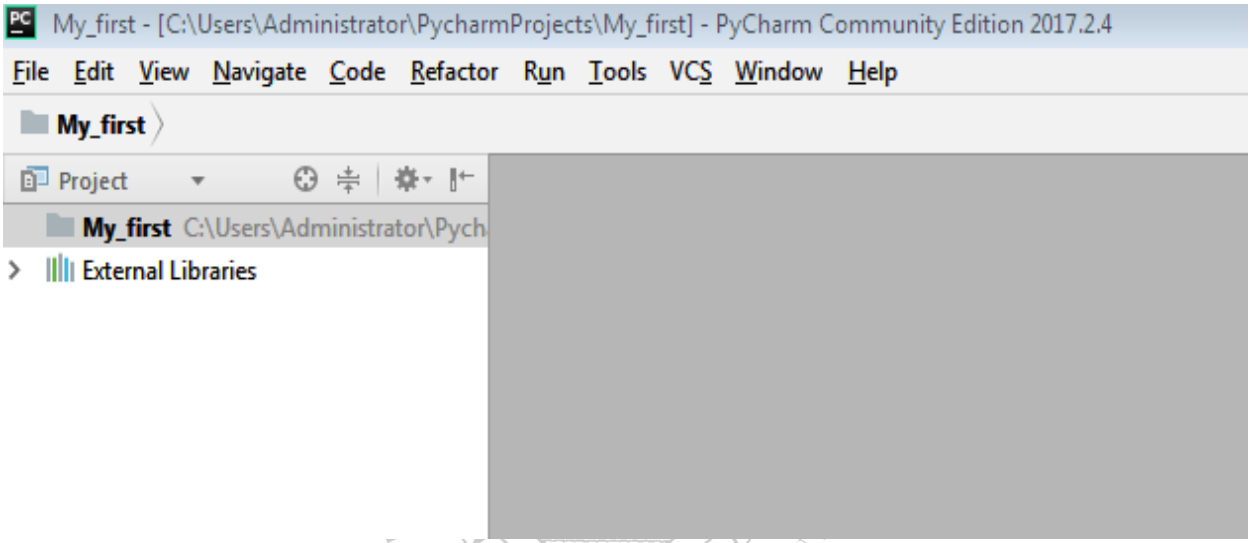


Fig. 8.1

3. Right click on the project (‘My_first’) and select the option New > Python File
4. Assign a name to that file (say First_script)
5. Write the code in the file and click on ‘Run’ to execute.

Program

```
#print in triangle
x=int(input("enter the number"))
for n in range(0,x):
    n +=1
    print ("*" *(0+n))

for n in range(-x,0):
    n +=1
    print ("*" *(0-n+1))
```

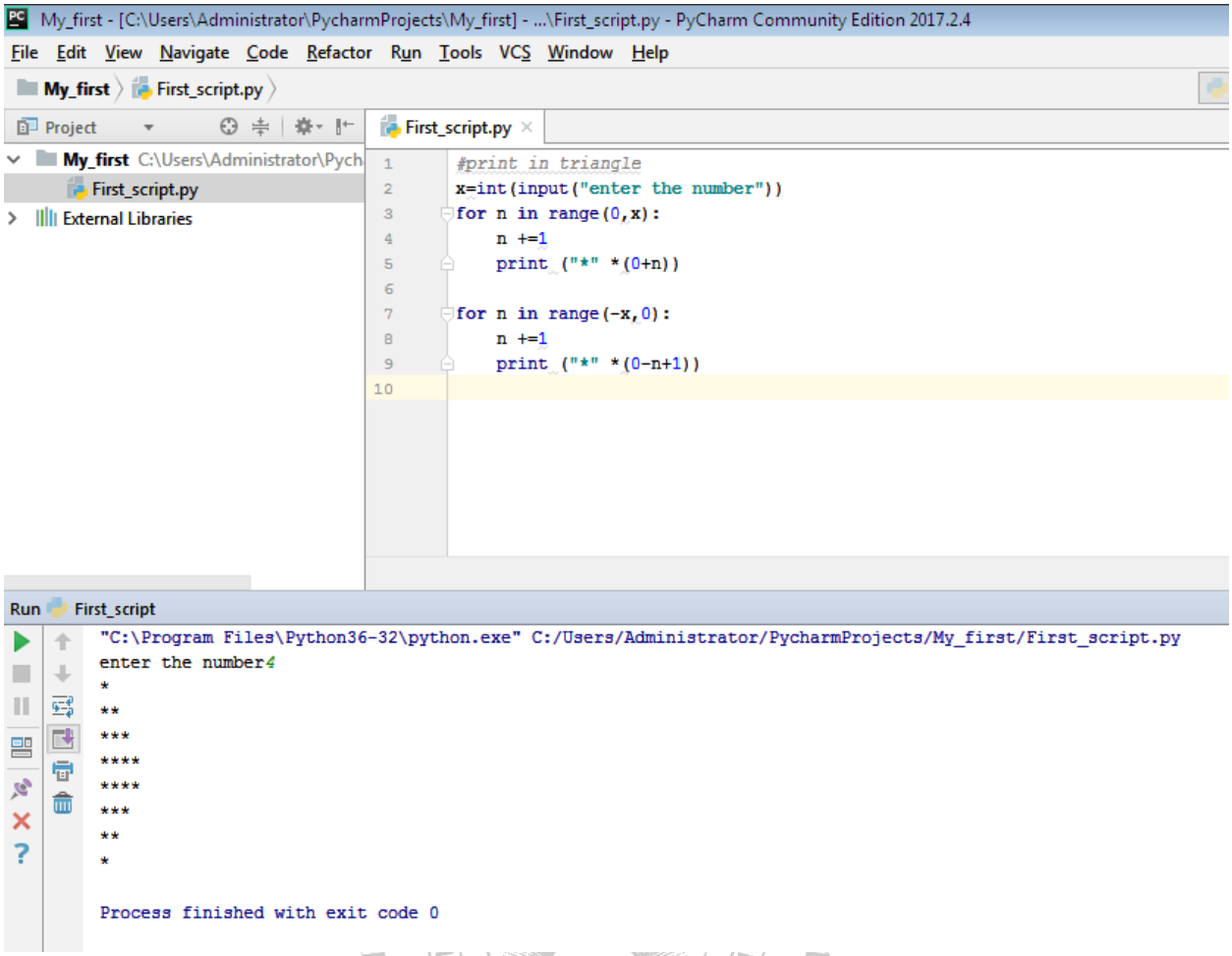


Fig. 8.2

EXCERCISE

1. Write a program to print the average of 5 integer values, entered by user using for loop.

Write the program here and attach the printout of output

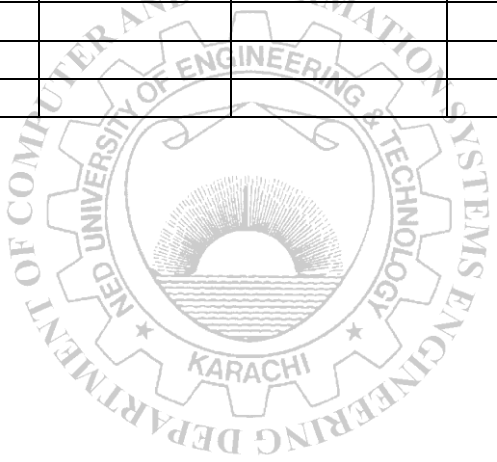
2. Explore debugging option on PyCharm using the program in Q-1 and describe it in own wording:

Attach the printout of output

3. Debug the following code and record your observations. Note the observations (values) for 5-iterations of while loop. Choose starting and ending range value accordingly.

```
r1 = int(input("Enter the starting range value?"))
r2 = int(input("Enter the ending range value?"))

num = r1 + 1
count = 0

while num < r2:

    res = num % 2

    if (num % 2) > 0:
        count += 1
    num += 1

print("Odd count: %d" % (count))
```

Variable	Iteration #1	Iteration # 2	Iteration #3	Iteration #4	Iteration #5
num					
count					
res					



Attach the printout of output

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Lab Session 09

Construct functions (Using PyCharm)

FUNCTION

A *function* is a groups of statements made to execute them more than once in a program. A function has a name. Functions can compute a result value and can have parameters that serve as function inputs which may differ each time when function is executed
Functions are used to:

- Reduce the size of code as it increases the code reusability
- Split a complex problem in to multiple modules (functions) to improve manageability

Scope Example

Let's step through a larger example that demonstrates scope ideas. Suppose we wrote the following code in a module file:

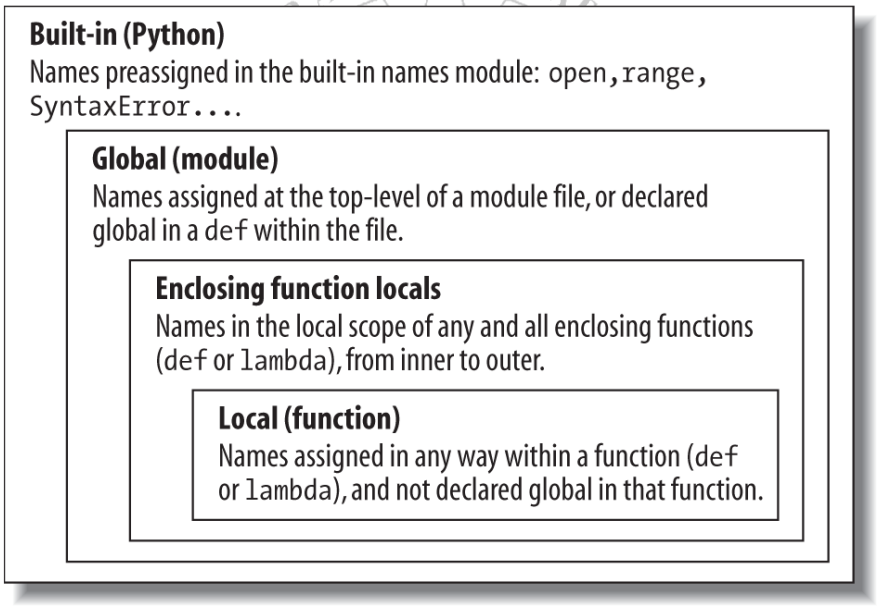


Fig. 9.1

Example-1

```
# Global scope

X = 99 # X and func assigned in module: global

def func(Y): # Y and Z assigned in function: locals

# Local scope

      Z = X + Y # X is a global
      return Z
```

```
func(1) # func in module: result=100
```

Example-2

```
X = 88 # Global X
def func():
    global X
    X = 99 # Global X: outside def
func()
print(X) # Prints 99
```

EXCERCISE

1. Develop a simple calculator (using functions) that defines addition, subtraction, multiplication and division operation. User selects the operation and provides the operands then output will be generated.

Write the program here and attach the printout of output

2. Debug the program of Q-1 to show the assignment of operands to variables and selection of operator through 'debug control' window through single stepping, over and out options separately.

Write down the difference in execution observed in three debugging ways, also attach the printout here:

3. Debug the following code. Observe the output. Is the global variable changing? If not how can global variable be altered?

```
name = 'xyz'

def change_name(new_name):
    name = new_name

print(name)

change_name('abc')

print(name)
```

4. Construct an outer function "`out_circle`" that takes radius '`r1`' of an outer circle as argument and calculates its area. Also construct an inner function "`in_circle`" that calculates it's circumference with a smaller radius. Inner function should be enclosed within the outer function.

Write the program and attach the output here

Lab Session 10

Construct recursive functions (Using PyCharm)

RECURSION

Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Termination condition

A recursive function has to terminate to be used in a program. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can lead to an infinite loop, if the base case is not met in the calls.

Example:
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1
Replacing the calculated values gives us the following expression
4! = 4 * 3 * 2 * 1

Generally we can say: Recursion is a method where the solution to a problem is based on solving smaller instances of the same problem.

Example

```
def mysum(L):
      if not L:
            return 0
      else:
            return L[0] + mysum(L[1:]) # Call mysum recursively
```

EXCERCISE

1. Develop and debug the Fibonacci series till user defined limit.

Write the program here and attach the printout of output

2. Generate the sum of n (user defined) natural number through recursive function. Also debug the code.

Write the program and attach output here

3. Develop and debug the recursive function to find the factorial of a number. The number should be taken as input from user.

Write the program and attach output here

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI

Lab Session 11

Construct generator functions (Using PyCharm)

GENERATOR FUNCTIONS

A Python generator is a function that produces a sequence of results. It works by maintaining its local state, so that the function can resume again exactly where it left off when called subsequent times. Thus, you can think of a generator as something like a powerful iterator.

The state of the function is maintained through the use of the keyword `yield`, which has the following syntax:

```
yield[expression_list]
```

How do Python Generators Work?

In order to understand how generators work, let's use the simple example below:

Example-1

```
def numberGenerator(n):
      number = 0
      while number < n:
            yield number
            number + = 1
myGenerator = numberGenerator(3)

print(next(myGenerator))
print(next(myGenerator))
print(next(myGenerator))
```

The code above defines a generator named `numberGenerator`, which receives a value `n` as an argument, and then defines and uses it as the limit value in a while loop. In addition, it defines a variable named number and assigns the value zero to it.

Calling the "instantiated" generator (`myGenerator`) with the `next()` method runs the generator code until the first yield statement, which returns 1 in this case.

Even after returning a value to us, the function then keeps the value of the variable `number` for the next time the function is called and increases its value by one. So the next time this function is called, it will pick up right where it left off.

Calling the function two more times, provides us with the next 2 numbers in the sequence, as seen below:

Output
0
1
2

If we were to have called this generator again, we would have received a `StopIteration` exception since it had completed and returned from its internal while loop.

In conclusion, g*enerator functions* are coded as normal def statements, but use yield statements to return results one at a time, suspending and resuming their state between each

Example-2

```
def gensquares(N):
      for i in range(N):
            yield i ** 2 # Resume here later
```

EXCERCISE

1. Differentiate between recursive and generator functions.

2. Develop a generator function to produce Fibonacci series till n (defined by user)

Write the program and attach output here

3. Develop a generator function to produce prime number till n (defined by user)

Write the program and attach output here

Lab Session 12

Practice implementation of tuples on Project Jupyter

TUPLES

Tuples construct simple groups of objects. They work exactly like lists, except that tuples can't be changed in place. (They are immutable)

Tuples should generally store values that are somehow different from each other. For example, we would not put three stock symbols in a tuple, but we might create a tuple of stock symbol, current price, high, and low for the day. The primary purpose of a tuple is to aggregate different pieces of data together into one container.

We can create a tuple by separating the values with a comma. Usually tuples are wrapped in parentheses to make them easy to read and to group them from other parts of an expression, but this is not always mandatory. The following two assignments are identical (they record a stock, the current price, the high, and the low for a rather profitable company):

```
>>> stock = "GOOD", 613.30, 625.86, 610.50
>>> stock2 = ("GOOD", 613.30, 625.86, 610.50)
```

Operations on Tuples

```
>>> t=()                          #An empty Tuple
>>> (1, 2) + (3, 4)          # Concatenation
(1, 2, 3, 4)
>>> (1, 2) * 4                 # Repetition
(1, 2, 1, 2, 1, 2, 1, 2)
>>> T = (1, 2, 3, 4)        # Indexing, slicing
>>> T[0], T[1:3]
(1, (2, 3))
>>> T = ('cc', 'aa', 'dd', 'bb')
>>> tmp = list(T)                 #Converting tuple into list
>>> tmp
['cc', 'aa', 'dd', 'bb']
>>> tmp.sort()                    #Sorting list
>>> tmp
['aa', 'bb', 'cc', 'dd']
>>> T = tuple(tmp)                #Converting list into tuple
>>> T
('aa', 'bb', 'cc', 'dd')
>>> sorted(T)                     #Sorting Tuple
['aa', 'bb', 'cc', 'dd']
```

JUPYTER NOTEBOOK

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access or can be accessed through the internet.

Classic notebook introduces the basic features of IPython.

IPython provides a rich toolkit to help you make the most of using Python interactively. Its main components are:

- A powerful interactive Python shell
- A Jupyter kernel to work with Python code in Jupyter notebooks and other interactive frontends

Access Notebook Server

Visit https://jupyter.org/try. Select 'Try Classic Notebook'. The following webpage appears.

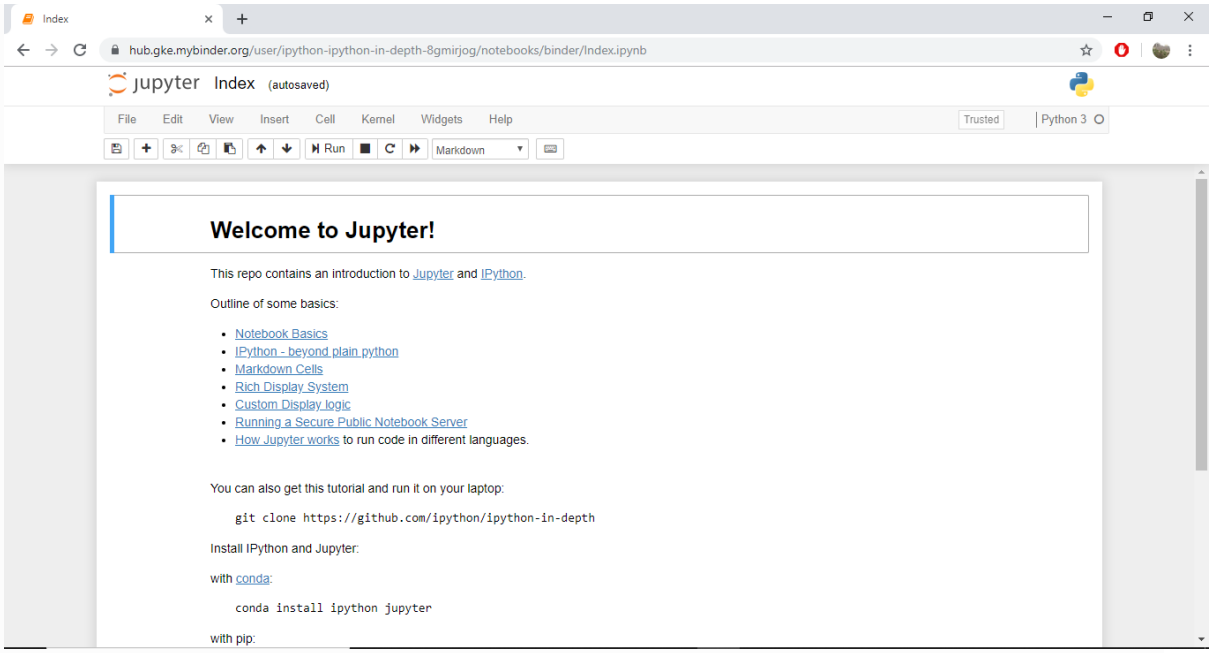


Fig 14.1

Creating a Notebook

Click on *File>New Notebook>Python3*, Your web page should now look like this:

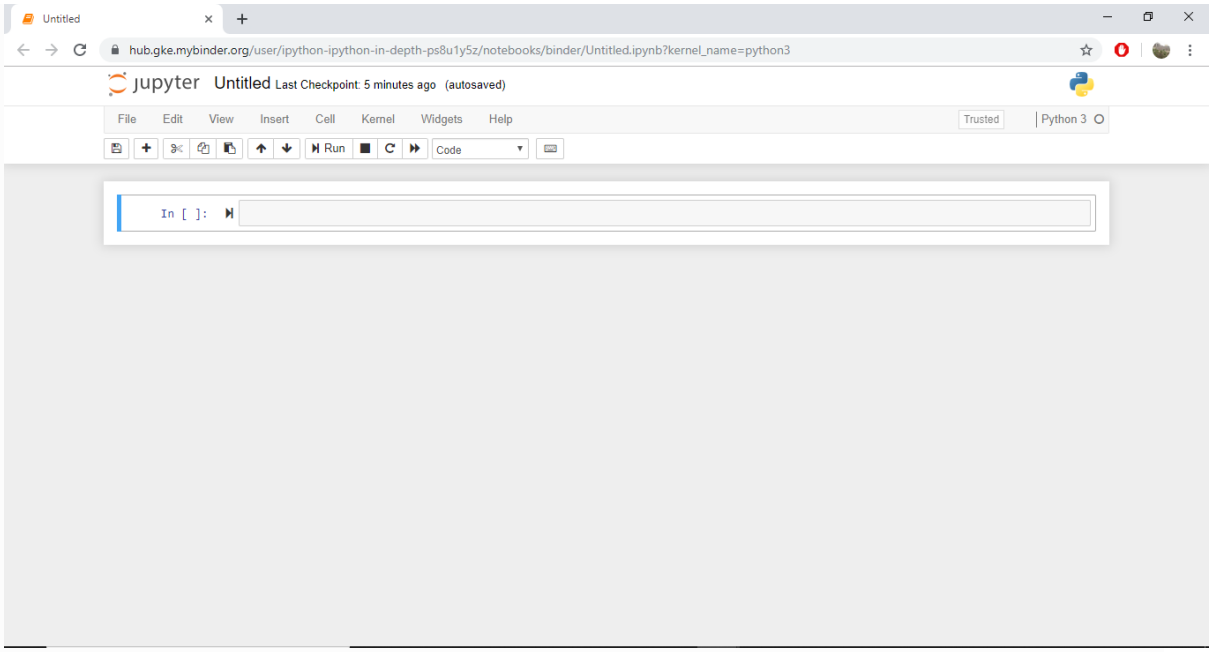


Fig 14.2

Naming

Move your mouse over the word *Untitled* and click on the text. You should now see an in-browser dialog titled *Rename Notebook*. Rename this one to Hello Jupyter*:*

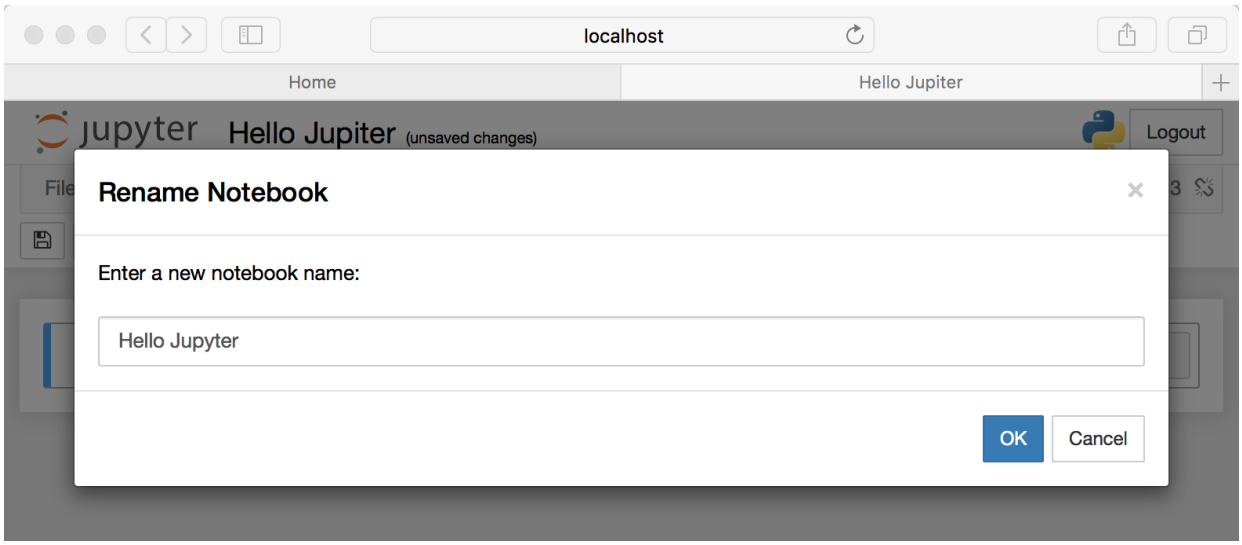


Fig 14.3

Running Cells

A Notebook's cell defaults to using code whenever you first create one, and that cell uses the kernel that you chose when you started your Notebook.

In this case, you started with Python 3 as your kernel, so that means you can write Python code in your code cells. Since your initial Notebook has only one empty cell in it, the Notebook can't really do anything.

Thus, to verify that everything is working as it should, you can add some Python code to the cell and try running its contents.

Let's try adding the following code to that cell:

```
print('Hello Jupyter!')
```

To execute a cell, just select the cell and click the *Run* button that is in the row of buttons along the top.

Output

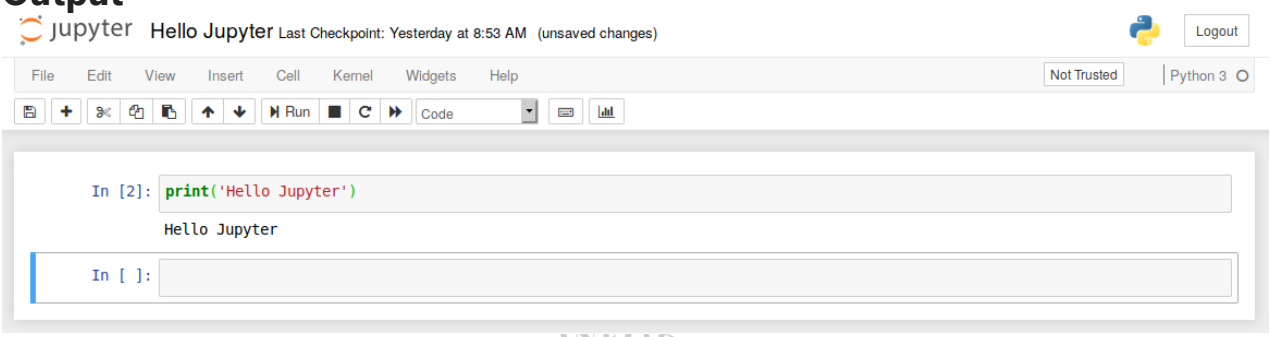


Fig 14.1

If there are multiple cells in Notebook, and the cells are run in order, you can share your variables and imports across cells. This makes it easy to separate out your code into logical chunks without needing to reimport libraries or recreate variables or functions in every cell.

When you run a cell, you will notice that there are some square braces next to the word *In* to the left of the cell. The square braces will auto fill with a number that indicates the order that you ran the cells. For example, if you open a fresh Notebook and run the first cell at the top of the Notebook, the square braces will fill with the number *1*.

The Menus

The Jupyter Notebook has several menus that help to interact with the Notebook. The menu runs along the top of the Notebook. Here is a list of the current menus:

- *File*
- *Edit*
- *View*
- *Insert*
- *Cell*
- *Kernel*
- *Widgets*
- *Help*

EXCERCISE

1. Develop a program to replace last element of all tuples in a list.

Write the program and attach output here

2. Write commands to perform operations on my_tuple = ('p','r','o','g','r','a','m','m','i','n','g') to convert it into :

('r', 'o', 'g') Command=

('p', 'r', 'o') Command=

('m', 'i', 'n', 'g') Command=

('p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g') Command=

3. Develop a script that takes a tuple from user and sorts it in reverse order.

Write the program and attach output here

Lab Session 13

Practice File Handling to read and write data (Using PyCharm)

FILES

Files are named storage compartments on computer that are managed by operating system.

Here mode can be typically the string 'r' to open for text input (the default), 'w' to create and open for text output, or 'a' to open for appending text to the end.

Open a text file
```
fh = open("hello.txt", "r")
```

Read a text file
```
fh = open("hello.txt","r")
print (fh.read())
```

Read one line at a time
```
fh = open("hello".txt", "r")
print (fh.readline())
```

Read a list of lines
```
fh = open("hello.txt.", "r")
print (fh.readlines())
```

Write to a file
```
fh = open("hello.txt","w")
write("Hello World")
fh.close()
```

Write a list of lines to a file
```
fh = open("hello.txt", "w")
lines_of_text = ["a line of text", "another line of text", "a third
line"]
fh.writelines(lines_of_text)
fh.close()
```

Append to a file
```
fh = open("Hello.txt", "a")
write("Hello World again")
fh.close()
```

Close a file
```
fh = open("hello.txt", "r")
print fh.read()
fh.close()
```

Sample Program (to read a file)
```
f = open("test.txt",'r')     # open file in current directory
print (f.read(5))
```

```
print('\n')
print (f.read(8))
print (f.readlines(1))
print (f.readlines())
```

Output

```
this
is first
[' file for python\n']
['this is the second line\n', 'this is the third line of first
python file']
```

Sample Program (to write a file)

```
file = open('test.txt', 'w')

file.write('This is a first script')
file.write('To add more lines in a file.')

file.close()
```

EXCERCISE

1. Develop a program to read a file in a remote directory with 'for loop'

Write your program here

2. Develop a script that prints the words of file separated by commas.

Write your program here

3. Develop a script to find the longest word in the file.

Write your program here

Lab Session 14

Complex Engineering Activity

Problem statement

Complex problem solving attributes covered

Task description

Deliverables

Grading rubric

Solution

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering

Course Code and Title: CS-115 Computer Programming

Laboratory Session No. _____ Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
To what extent is the student familiar with the scripting/ programming interface?	The student is unfamiliar with the interface.	The student is familiar with few features of the interface.	The student is familiar with many features of the interface.	The student is proficient with the interface.

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering

Course Code and Title: CS-115 Computer Programming

Laboratory Session No. _____ Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
To what extent is the student familiar with the scripting/ programming interface?	The student is unfamiliar with the interface.	The student is familiar with few features of the interface.	The student is familiar with many features of the interface.	The student is proficient with the interface.

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering

Course Code and Title: CS-115 Computer Programming

Laboratory Session No. _____ Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
To what extent is the student familiar with the scripting/ programming interface?	The student is unfamiliar with the interface.	The student is familiar with few features of the interface.	The student is familiar with many features of the interface.	The student is proficient with the interface.

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

DEPARTMENT OF COMPUTER AND INFORMATION SYSTEMS ENGINEERING
NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI