R18

KEY **Code No: 153CD**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B. Tech II Year I Semester Examinations, March - 2022
PYTHON PROGRAMMING
(Common to CSBS, CSIT, ITE, CSE(SE), CSE(CS), CSE(AIML), CSE(DS), CSE(IOT), CSEN)

1 a) **Summarize the built in functions in Python with appropriate examples.** 8M

A function is a group of statements that exist within a program for the purpose of performing a specific task

The Python interpreter has a number of functions built into it that are always available.

Built-in Functions

abs() dict() help() min() setattr()
all() dir() hex() next() slice()
any() divmod() id() object() sorted()
ascii() enumerate() input() oct() staticmethod()
bin() eval() int() open() str()
bool() exec() isinstance() ord() sum()
bytearray() filter() issubclass() pow() super()
bytes() float() iter() print() tuple()
callable() format() len() property() type()
chr() frozenset() list() range() vars()
classmethod() getattr() locals() repr() zip()
compile() globals() map() reversed() __import__()
complex() hasattr() max() round()
delattr() hash() memoryview() set()

Student can write any 8 examples each will get one mark

1 b) **Compare Lists and Tuples with an example.** 7M

List : A List is an ordered collection of items (which may be of same or different types) separated by comma and enclosed in square brackets.

Example:

```
In [1]:
L1=[10,25.5,3+2j,"Hello"]
L1
Out[1]:
[10, 25.5, (3+2j), 'Hello']
```

Tuple: Tuple looks similar to list. The only difference is that comma separated items of same or different type are enclosed in parentheses. Individual items follow zero based index, as in list or string.

```
In [3]:
T1=(10,25.5,3+2j,"Hello")
T1
```

Out[3]:
(10, 25.5, (3+2j), 'Hello')
In [4]:
T1[1]
Out[4]:
25.5
Difference between List and Tuple:
The obvious difference is the use of square brackets [] in List and parentheses () in tuple as enclosures. However, the important difference is that List as a mutable object and Tuple is an immutable object.

If contents of an object can be modified in place, after it has been instantiated, is a mutable object. On the other hand, any operation on immutable object that tries to modify its contents is prohibited.

In above example, any item in the list L1 can be assigned with different value. Let us change value of item at index=2 from 3+2j to 1.22E-5

In [5]:
L1[2]=1.22E-5
L1
Out[5]:
[10, 25.5, 1.22e-05, 'Hello']
The built-in List class has different methods that allow various operations on List object (such as insertion, deletion, sorting etc)

However, any such operation is not possible with Tuple object. If we try to modify T1 by changing value of item at index=2 to 1.22E-5, TypeError exception is raised.

In [6]:
T1[2]=1.22E-5
T1

TypeError Traceback (most recent call last)
<ipython-input-6-18a8ef5d3740> in <module>()
----> 1 T1[2]=1.22E-5
2 T1
TypeError: 'tuple' object does not support item assignment

2 a) **List out and explain various types of operators used in Python.** 7M

Operators:
• Operators are the tokens that trigger some computation / action when

applied to variables and other objects in an expression.
Operands:
• Variables and Objects on which operations are applied by operators are called operands.
Types of Operators in Python: Python has following types of operators
1. Unary Operators 6. Relational Operators
2. Binary Operators 7. Assignment Operators
3. Bitwise Operators 8. Logical Operators
4. Shift Operators 9. Membership Operators
5. Identity Operators
UNARY OPERATORS
• Unary Operators are those operators that require one operand to operate upon. Following are some unary operators
Unary + Operator:
• This operators precedes an operand.
• Operand must be of arithmetic type.
• The result is the value of the argument.
Example: a= 5 then +a means 5
a=-5 then +a means -5
Operators Symbol Meaning
+ Unary Plus
- Unary Minus
~ Bitwise Complement not Logical negation
BINARY OPERATORS
• Binary Operators are those operators that requires two operands to operate upon. Following are some binary operators in Python
Operators Symbol Meaning
+ Addition
- Subtraction
* Multiplication
/ Division
% Remainder / Modulus
** Exponent (Raise to Power)
// Floor Division
RELATIONAL OPERATORS
• The relational operator determine the relation among different operands.
• Python provides six relational operators for comparing values thus also called comparison operators.
• If the comparison is true, the relational expressions results in Boolean value True and to Boolean value false if the comparison is false.
• These operators as below:
< Less than

<= Less than or Equal to
> Greater than
>= Greater than or Equal to
== Equal to
!= not Equal to
IDENTITY OPERATORS
• There is two identity operators in Python. is and is not.
• The identity operators are used to check if both the operands reference the same memory object.
• It means the identity operators compares the memory location of two objects and return True of False accordingly.
Operator Usage Description
is a is b Return True if a and b both pointing to same memory location i.e. same object otherwise False.
is not a is not b Returns True if a and b both pointing to different memory locations i.e. different objects otherwise False.
LOGICAL OPERATORS
Let us Take third Example related to floating Point Literals
>>>K=3.5
>>>L=float(input("Enter a Real number:- "))
Enter the same value i.e.3.5 from keyboard for variable L.
>>>K==L
output
True
>>>K is L Different Output
output
False
Reasons behind Returning False by Identity Operator is that Python creates two different objects for the following cases
1. Input of string from the console / Keyboard
2. Writing Integers with many digits (big integers)

2 b) **Differentiate between List Slicing and List Mutability with examples.** 8M

List slicing refers to accessing a specific portion or a subset of the list for some operation while the orginal list remains unaffected. The slicing operator in python can take 3 parameters out of which 2 are optional depending on the requirement.

Syntax of list slicing:

list_name[start:stop:steps]

The start parameter is a mandatory parameter, whereas the stop and steps are both optional parameters.

The start represents the index from where the list slicing is supposed to begin. Its default value is 0, i.e. it begins from index 0.

The stop represents the last index up to which the list slicing will go on. Its default value is (length(list)-1) or the index of last element in the list.
Example:

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = ['x', 'y']
print(alist)
```

Mutability:

Mutability is the ability for certain types of data to be changed without entirely recreating it. This is important for Python to run programs both quickly and efficiently. Mutability varies between different data types in Python. A list is mutable
Example:

```
>>> li = [1, 2, 3, 4]
>>> li.reverse()        # nothing gets returned
>>> li                  # changed!
[4, 3, 2, 1]
```

Student can write any related examples regaring slicing and mutability

3 a) **How to create a module? Give an example**. 8M

Modules:

A module is a file containing Python definitions and statements. The file name is the module name with the suffix.py appended. Within a module, the module‟s name (as a string) is available as the value of the global variable __name .
Example:

```
# Fibonacci numbers module
def fib(n): # write Fibonacci series up to n a, b = 0, 1
while b < n: print b,
a, b = b, a+b
def fib2(n): # return Fibonacci series up to n result = []
a, b = 0, 1
while b < n: result.append(b) a, b = b, a+b
return result
```

Using Module:

```
>>> import fibo
```

This does not enter the names of the functions defined in fibo directly in the

current symbol table; it only enters the module name fibo there. Using the module name we can access the functions:

```
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Student can write any example on modules

3 b) **Illustrate how to handle an exceptions in python.** 7M

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.

We can thus choose what operations to perform once we have caught the exception. Here is a simple example.

Syntax

```
try:
   #block of code

except Exception1:
   #block of code

except Exception2:
   #block of code

#other code
```

Example

```
try:
   a = int(input("Enter a:"))
   b = int(input("Enter b:"))
   c = a/b
   print("a/b = %d"%c)
except ZeroDivisionError:
   print("can't divide by zero")
   print(Exception)
finally:
         print("in finally block")
```

4 a) **How to create a text file and write content in to the text file? Explain with a python script.** 7M

For writing to a file, we first need to open it in write or append mode. If we open an existing file in write mode, the previous data will be erased, and the file object will be positioned at the beginning of the file. On the other hand, in append mode, new data will be added at the end of the previous data as the file object is at the end of the file. After opening the file, we can use the following methods to write data in the file.

• write() - for writing a single string

• writeline() - for writing a sequence of strings

write() method takes a string as an argument and writes it to the text file. It returns the number of characters being written on single execution of the write() method.

Also, we need to add a newline character (\n) at the end of every sentence to mark the end of line.

Example:

```
>>> myobject=open("myfile.txt",'w')
>>> myobject.write("Hey I have started #using files in Python\n")
>>> myobject.close()
```

On execution, write() returns the number of characters written on to the file.

The writelines() method

This method is used to write multiple strings to a file.We need to pass an iterable object like lists, tuple, etc. containing strings to the writelines() method. Unlike write(), the writelines() method does not return thenumber of characters written in the file.

The following code explains the use of writelines().

```
>>> myobject=open("myfile.txt",'w')
>>> lines = ["Hello everyone\n", "Writing #multiline strings\n", "This is the
#third line"]
>>> myobject.writelines(lines)
>>>myobject.close()
```

4 b) **What are the various file opening modes in python?** 8M

Python file modes

- r for reading – The file pointer is placed at the beginning of the file. This is the default mode.
- r+ Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
- w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.

- rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file.
- rb+ Opens a file for both reading and writing in binary format.
- wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, it creates a new file for reading and writing.
- a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
- ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
- x open for exclusive creation, failing if the file already exists (Python 3)

5 a) **Discuss about various types of regular expressions in python.** 8M

Regular expressions are a powerful language for matching text patterns. The Python "re" module provides regular expression

Basic Patterns

The power of regular expressions is that they can specify patterns, not just fixed characters. Here are the most basic patterns which match single chars:

- a, X, 9, < -- ordinary characters just match themselves exactly. The meta-characters which do not match themselves because they have special meanings are: . ^ $ * + ? { [ ] \ | ( ) (details below)
- . (a period) -- matches any single character except newline '\n'
- \w -- (lowercase w) matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]. Note that although "word" is the mnemonic for this, it only matches a single word char, not a whole word. \W (upper case W) matches any non-word character.
- \b -- boundary between word and non-word
- \s -- (lowercase s) matches a single whitespace character -- space, newline, return, tab, form [\n\r\t\f]. \S (upper case S) matches any non-whitespace character.
- \t, \n, \r -- tab, newline, return
- \d -- decimal digit [0-9] (some older regex utilities do not support \d, but they all support \w and \s)

- ^ = start, $ = end -- match the start or end of the string
- \ -- inhibit the "specialness" of a character. So, for example, use \. to match a period or \\ to match a slash. If you are unsure if a character has special meaning, such as '@', you can put a slash in front of it, \@, to make sure it is treated just as a character.

Method used in Regular Expressions:

- match(){does it match the beginning of my string? Returns None or a match object
- search(){does it match anywhere in my string? Returns None or a match object
- findall(){does it match anywhere in my string? Returns a list of strings (or an empty list)
- sub() Replace one or many matches in the string.
- Split() Returns a list in which the string has been split in each match.

Example:

```
import re
string="Simple is better than complex."
obj=re.search(r"better", string)
if obj!=None:
   print("yes",obj.start(),obj.end())
else:
   print("no")
```

Student can write any other examples using Regular Expressions

5 b) **Write about Global Interpreter Lock.** 7M

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

This means that only one thread can be in a state of execution at any point in time. The impact of the GIL isn't visible to developers who execute single-threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.

Since the GIL allows only one thread to execute at a time even in a multi-threaded architecture with more than one CPU core, the GIL has gained a reputation as an "infamous" feature of Python.

6 a) **Discuss about all the methods in Thread module.** 7M

The methods provided by the Thread class are as follows −

- run() − The run() method is the entry point for a thread.

- start() − The start() method starts a thread by calling the run method.
- join([time]) − The join() waits for threads to terminate.
- isAlive() − The isAlive() method checks whether a thread is still executing.
- getName() − The getName() method returns the name of a thread.
- setName() − The setName() method sets the name of a thread.
- tactiveCount() − Returns the number of thread objects that are active.
- currentThread() − Returns the number of thread objects in the caller's thread control.
- enumerate() − Returns a list of all thread objects that are currently active.

6 b) **Write a python script for Inter process communication.** 8M

```
import multiprocessing
result = []
def square_list(mylist):
        global result
        for num in mylist:
        result.append(num * num)
    print("Result(in process p1): {}".format(result))
if __name__ == "__main__":
        mylist = [1,2,3,4]
    p1 = multiprocessing.Process(target=square_list, args=(mylist,))
    p1.start()
    p1.join()
    print("Result(in main program): {}".format(result))
```

```
import multiprocessing
def square_list(mylist, result, square_sum):
    for idx, num in enumerate(mylist):
        result[idx] = num * num
    square_sum.value = sum(result)
    print("Result(in process p1): {}".format(result[:]))
    print("Sum of squares(in process p1): {}".format(square_sum.value))
if __name__ == "__main__":
    mylist = [1,2,3,4]
    result = multiprocessing.Array('i', 4)
    square_sum = multiprocessing.Value('i')
    p1 = multiprocessing.Process(target=square_list, args=(mylist, result,
square_sum))
    p1.start()
    p1.join()
    print("Result(in main program): {}".format(result[:]))
    print("Sum of squares(in main program): {}".format(square_sum.value))
```

Student can write any other examples similar like above

7 a) **How to build a CGI application in Python?** 8 M

CGI Stands for Common Gateway Interface

CGI is a set of standards that defines a standard way of passing information or web-user requests to an application program and getting data back to forward it to users. It is the exchange of information between the web server and a custom script. When the users requested the web-page, the server sends the requested web-page. The web server usually passes the information to all application programs that process data and sends back an acknowledged message; this technique of passing data back-and-forth between server and application is the Common Gateway Interface.

CGI support modules

Python's standard library consists of two module for CGI support. The cgi module defines number of utilities to be used by Python CGI script. The cgitb module is a traceback manager for CGI scripts. Normally both modules are imported in a Python script, enabling the traceback feature.

```
import cgi, cgitb
```

Example:

```
<form action = "/cgi-bin/hello_get.py" method = "post">
First Name: <input type = "text" name = "first_name">  <br />
Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

```
#!"C:\Users\Pavan\AppData\Local\Programs\Python\Python310\python.exe"
print ("Content-type:text/html\r\n\r\n")
print()
import cgi,cgitb
cgitb.enable() #for debugging
form = cgi.FieldStorage()
fname = form.getvalue('first_name')
sname=form.getvalue('last_name')
print ("<html>")
print ("<head>")
print ("<title>My First CGI-Program </title>")
print ("<head>")
print ("<body>")
print("FirstName of the user is:",fname)
print("SName of the user is:",sname)
print ("</body>")
print ("</html>")
```

```
----------------------------
<form action = "/cgi-bin/hello_get1.py" method = "get">
no1 <input type = "text" name = "n1">  <br />

no2 <input type = "text" name = "n2" />
<input type = "submit" value = "+" name="sub" />
<input type = "submit" value = "-" name="sub" />
<input type = "submit" value = "*" name="sub" />
<input type = "submit" value = "/" name="sub" />
</form>
---------------------------------
#!"C:\Users\Pavan\AppData\Local\Programs\Python\Python310\python.exe"
print ("Content-type:text/html\r\n\r\n")
print()

import cgi,cgitb
cgitb.enable() #for debugging
form = cgi.FieldStorage()
sn1 = form.getvalue('n1')
sn2 = form.getvalue('n2')
a=int(sn1)
b=int(sn2)
submitval=form.getvalue('sub')
print ("<html>")
print ("<body>")

if(submitval=="+"):
        print("Output is:",(a+b))
elif(submitval=="-"):
        print(":",(a-b))
elif(submitval=="*"):
        print("output:",(a*b))
else:
        print("output:",(a/b))

print ("</body>")
print ("</html>")
```

Student can write any other examples similar like above

7 b) **Write a program to demonstrate a group of check button widgets**. 7M

```
from tkinter import *

root = Tk()
```

```
root.geometry("300x300")
w = Label(root, text ='JNTUH', fg="Blue",font = "100")
w.pack()
Checkbutton1 = IntVar()
Checkbutton2 = IntVar()
Checkbutton3 = IntVar()
Button1 = Checkbutton(root, text = "Homepage",
                                        variable = Checkbutton1,
                                        onvalue = 1,
                                        offvalue = 0,
                                        height = 2,
                                        width = 10)

Button2 = Checkbutton(root, text = "Tutorials",
                                        variable = Checkbutton2,
                                        onvalue = 1,
                                        offvalue = 0,
                                        height = 2,
                                        width = 10)

Button3 = Checkbutton(root, text = "Contactus",
                                        variable = Checkbutton3,
                                        onvalue = 1,
                                        offvalue = 0,
                                        height = 2,
                                        width = 10)
Button1.pack()
Button2.pack()
Button3.pack()
mainloop()
```

8 a) **Write a python script to create a table, insert and display data from the database**. 8M

```
import mysql.connector
mydb = mysql.connector.connect( host="localhost", user="root", passwd="",
database="eceab")
print("data base connected")
mycursor=mydb.cursor()
#table createion
mycursor.execute("CREATE TABLE sss(NO INT(20),NAME
VARCHAR(20))")
print("TABLE created ");
#insert data
```

```
sql="INSERT INTO tb1 (NO, NAME) VALUES ('JNTUH','python')";
mycursor.execute(sql)
print("row created ");
sql="INSERT INTO tb1 (NO, NAME) VALUES ('OU','HYD')";
mycursor.execute(sql)
print("row created ");
mydb.commit()
#-display data
mycursor.execute("SELECT * FROM TB1")
myresult=mycursor.fetchall()
for row in myresult:
   print(row)
```

8 b) **Explain various keys and constraints used in SQLite while creating relations.** 7M

Following are commonly used constraints available in SQLite.

- **NOT NULL Constraint** − Ensures that a column cannot have NULL value.
- **DEFAULT Constraint** − Provides a default value for a column when none is specified.
- **UNIQUE Constraint** − Ensures that all values in a column are different.
- **PRIMARY Key** − Uniquely identifies each row/record in a database table.
- **CHECK Constraint** − Ensures that all values in a column satisfies certain conditions.

Examples:

SQLite NOT NULL constraint

A column with a NOT NULL constraint cannot have NULL values.

➔ CREATE TABLE People(Id INTEGER, LastName TEXT NOT NULL, FirstName TEXT NOT NULL, City TEXT);

SQLite Default constraint

The DEFAULT constraint inserts a default value into the column if no value is available.

CREATE TABLE Hotels(Id INTEGER PRIMARY KEY, Name TEXT, City TEXT DEFAULT 'not available');

SQLite UNIQUE constraint

The UNIQUE constraint ensures that all data are unique in a column.

CREATE TABLE Brands(Id INTEGER, BrandName TEXT UNIQUE);

SQLite Primary key constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

CREATE TABLE Brands(Id INTEGER PRIMARY KEY, BrandName TEXT);

SQLite Check constraint

A CHECK clause imposes a validity constraint on a relational database's data.

```
CREATE TABLE Orders(Id INTEGER PRIMARY KEY, OrderPrice
INTEGER CHECK(OrderPrice>0),Customer TEXT);
```