CBCS Scheme

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

16MCA21

Second Semester MCA Degree Examination, June/July 2017
Python Programming

Time: 3 hrs. Max. Marks: 80

Note: ***Answer FIVE full questions, choosing one full question from each module.***

Module-1

1 a. How does a computer run a Python Program? Explain with a neat diagram. **(06 Marks)**

b. Explain and construct the memory model of variables in Python. **(06 Marks)**

c. List any four built – in string functions in Python and explain. **(04 Marks)**

OR

2 a. Predict the output of the following code and justify your answer :

```
City = "Bengaluru"
City [1] = City [8] = "e"
City [6] = "0"
Print (city).
```

(02 Marks)

b. Trace the function call and explain the memory model of the following code :

```
def f(x) :
    X = 2 * x
    return x
x = 1
x = f (x + 1).
```

(08 Marks)

c. Discuss the usage of the following with respect to the print ( ) function
i) sep argument ii) end argument iii) .format (arguments). **(06 Marks)**

Module-2

3 a. Predict the output of the following and justify your answer : **(04 Marks)**
i) not "False" ii) -17 % 10 iii) (212 – 32) * 5 / 9 iv) 3.5 // 1.3.

b. Write a Python program to find average of best two test marks out of three test marks. **(04 Marks)**

c. What are the two ways of importing a module? Which one is more beneficial? Explain. **(08 Marks)**

OR

4 a. Discuss the importance of docstring in testing the code semi – automatically using doctest. **(08 Marks)**

b. Write a Python program to find the roots of a quadratic equation. **(08 Marks)**

Module-3

5 a. Consider the list qty = [5, 4, 7, 3, 6, 2, 1] and write the Python code to perform the following operation without using built-in methods :
i) Insert an element 9 at the beginning of the list ii) Insert an element 8 at the end of the list iii) Insert an element 8 at the index position 3 of the list iv) Delete an element at the beginning of the list v) Delete an element at the end of the list vi) Delete an element at the index position 3 vii) Print the list in reverse order (end to start)
viii) Delete all the elements of the list. **(08 Marks)**

b. Write the Python program to check whether a given number is prime or not, using for – else statement. **(08 Marks)**

OR

6 a. Give any four differences between a list and a string in Python. **(04 Marks)**

b. Write a Python program to read a string with punctuations and print the same string without punctuations. **(08 Marks)**

c. What is a list of lists? Give an example along with its memory model. **(04 Marks)**

Module-4

7 a. How can we use 'with' statement while opening a text file? Explain. **(04 Marks)**

b. Consider the following two sets and write the Python code to perform following operations on them. **(04 Marks)**

i) Union ii) Difference iii) Symmetric difference iv) Intersection	Lows = 0,1, 2, 3, 4 Odds = 1, 3, 5, 7, 9

c. Write a Python program to read a word and print the number of letters, vowels and percentage of vowels in the word using a dictionary. **(08 Marks)**

OR

8 a. Store the following data in a list, in a set and in a dictionary. **(06 Marks)**

India	USA	UK	Japan
91	1	41	81

b. In what situations are the sets more useful than the lists? **(02 Marks)**

c. Write a Python program to read the contents of a text file and write into another. **(08 Marks)**

Module-5

9 a. Write short notes on : i) is instance ( ) ii) __ init __ ( ). **(04 Marks)**

b. With an example, discuss the different components of a tkinter program. **(06 Marks)**

c. Write an object oriented Python program to create two time objects : Current _ time and Bread _ time which contains bread baking time. Include addTime method to display the total time taken by the bread maker to prepare a bread. **(06 Marks)**

OR

10 a. What are the steps that Python follows while creating an object? **(03 Marks)**

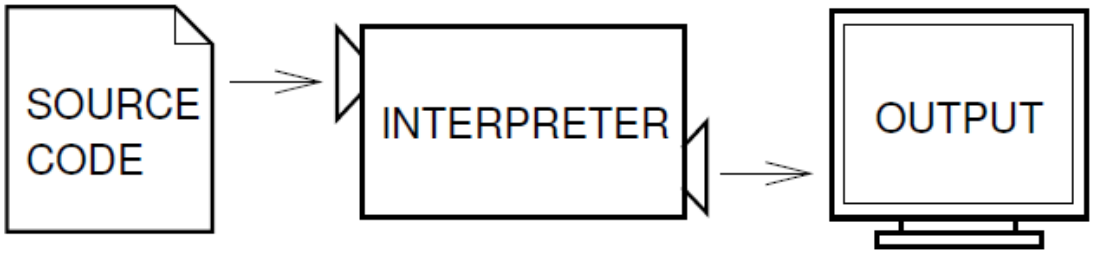
b. Explain MVC design with the help of tkinter program. **(08 Marks)**

c. Write a tkinter program to design a GUI window that has a lable of background color green and foreground color white. **(05 Marks)**

* * * * *

1a)**How does a computer run a Python program?Explain with a neat diagram.**

Python is an example of a high-level language.Two kinds of programs process high-level languages into low-level languages:interpreters and compilers. An interpreter reads a high-level program and
executes it, meaning that it does what the program says. It processes the programa little at a time, alternately reading lines and performing computations

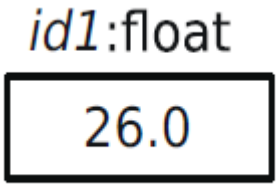


Python is considered an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter: commandline mode and script mode. In command-line mode, you type Python programsand the interpreter prints the result:

By convention, _les that contain Python programs have names that end with .py.
To execute the program, we have to tell the interpreter the name of the script:

```
$ python latoya.py
2
```

b)Memory model of variables in Python:
Every location in the computer's memory has a *memory address*, much like an address for a house on a street, that uniquely identifies that location.
We mark our memory addresses with an *id* prefix (short for *identifier*) so that they look different from integers: *id1*, *id2*, *id3*, and so on.
Memory model for the floating-point value 26.0



This picture shows the value 26.0 at the memory address *id1*. We will always show the type of the value as well—in this case, float. We will call this box an *object*: a value at a memory address with a type. During execution of a program, every value that Python keeps track of is stored inside an object in computer memory.
In our memory model, a variable contains the memory address of the object to which it refers:

In order to make the picture easier to interpret, we will usually draw arrows from variables to their objects.
Value 26.0 has the memory address *id1*.
• The object at the memory address *id1* has type float and the value 26.0.
• Variable degrees_celsius *contains* the memory address *id1*.
• Variable degrees_celsius *refers* to the value 26.0.
Whenever Python needs to know which value degree_celsius refers to, it looksat the object at the memory address that degree_celsius contains. In this example,that memory address is *id1*, so Python will use the value at the memory address *id1*, which is 26.0.

c)Python string functions:

Method	Description
str.capitalize()	Returns a copy of the string with the first letter capitalized and the rest lowercase
str.count(s)	Returns the number of nonoverlapping occurrences of s in the string
str.endswith(end)	Returns True iff the string ends with the characters in the end string—this is case sensitive.
str.find(s)	Returns the index of the first occurrence of s in the string, or -1 if s doesn't occur in the string—the first

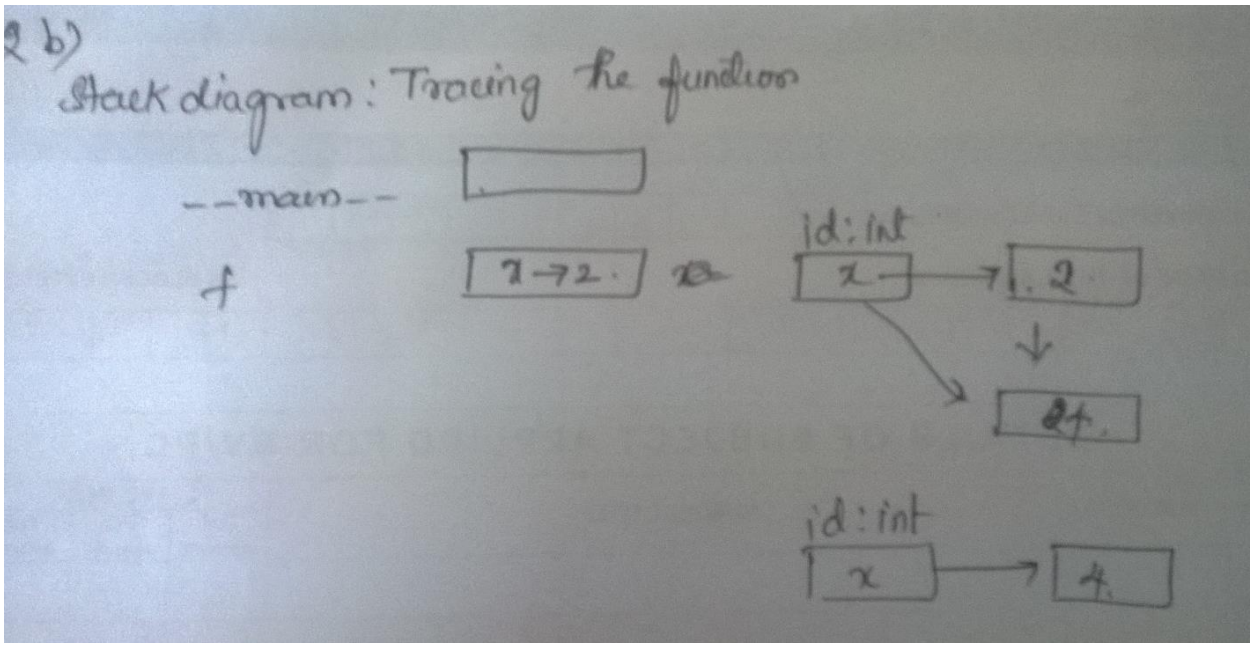
2)a) Predict the output of the following code and justify your answer:

```
City=”Bangaluru”
City[1]=City[8]=”e”
City[6]=”0”
Print(City)
```

OutPut: we get error as strings are immutable in python

b)Trace the function call and explain the memory model of the following code:

```
def f(x):
  X=2*x
  Return x
X=1
X=f(x+1)
```



c)Discuss the usage of the following with respect to the print() function
i)sep argument ii)end argument iii) .format(arguments)

i) sep argument:

The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice. The 'sep' parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Examples:

```
#code for disabling the softspace feature
print('G','F','G', sep='')

#for formatting a date
print('09','12','2016', sep='-')

#another example
print('pratik','gmail.com', sep='@')
```

Output:

```
GFG
09-12-2016
pratik@gmail.com
```

ii)end

> By default, a newline ("\n") is written after the last value in args. You may use this keywoard argument to specify a different line terminator, or no terminator at all.

The sep parameter when used with end parameter it produces awesome results. Some examples by combining the sep and end parameter.

```
print('G','F', sep='', end='')
print('G')
#n provides new line after printing the year
print('09','12', sep='-', end='-2016n')

print('prtk','agarwal', sep='', end='@')
print('geeksforgeeks')
```

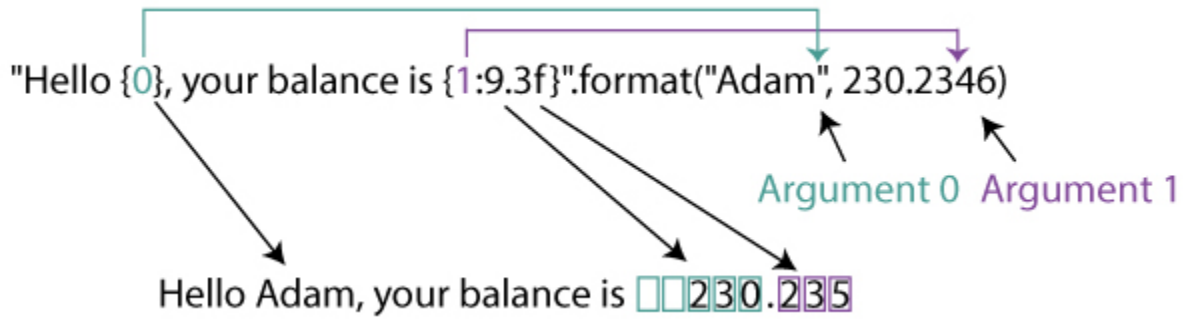
Output:

```
GFG
09-12-2016
prtkagarwal@geeksforgeeks
```

iii).format(arguments)

The string format() method formats the given string into a nicer output in Python.

The format() method returns the formatted string. The format() reads the type of arguments passed to it and formats it according to the format codes defined in the string.



Here, Argument 0 is a string "Adam" and Argument 1 is a floating number 230.2346.

The string `"Hello {0}, your balance is {1:9.3f}"` is the template string. This contains the format codes for formatting.

The curly braces are just placeholders for the arguments to be placed. In the above example, `{0}` is placeholder for `"Adam"` and `{1:9.3f}` is placeholder for `230.2346`.

Since the template string references format() arguments as {0} and {1}, the arguments are positional arguments. They both can also be referenced without the numbers as {} and Python internally converts them to numbers.

3a)Predict the output:

i)not "False" output:False

ii)-17%10 output:3

iii)(212-32)*5/9 output:100

iv)3.5//1.3 output:2

b**)Python program to find average of best two test marks out of three test marks:**

```
from array import *
my_array = list()
sum=0
print 'Enter 3 test marks : '
for i in range(0,3):
   n = raw_input("num :")
   my_array.append(int(n))
for i in range(0,3):
   sum=sum+my_array[i]
sum=sum-min(my_array[0],my_array[1],my_array[2])
avg=sum/2
print("avg of best two test marks is %d",avg)
```

c) Two ways of importing a module:

Python provides at least three different ways to import modules. You can use the *import* statement, the *from* statement, or the builtin *__import__* function.

- **import X** imports the module X, and creates a reference to that module in the current namespace. Or in other words, after you've run this statement, you can use *X.name* to refer to things defined in module X.

- **from X import *** imports the module X, and creates references in the current namespace to all *public* objects defined by that module (that is, everything that doesn't have a name starting with "_"). Or in other words, after you've run this statement, you can simply use a plain *name* to refer to things defined in module X. But X itself is not defined, so *X.name*doesn't work. And if *name* was already defined, it is replaced by the new version. And if *name* in X is changed to point to some other object, your module won't notice.

- **from X import a, b, c** imports the module X, and creates references in the current namespace to the given objects. Or in other words, you can now use *a* and *b* and *c* in your program.

- Finally, **X = __import__('X')** works like **import X**, with the difference that you 1) pass the module name as a string, and 2) explicitly assign it to a variable in your current namespace.

    - First is the namespace. Importing a function into the global namespace risks name collisions.

    - Second isn't that relevant to standard modules, but significant for you own modules, especially during development. It's the option to reload() a module.
    - we are importing only the function that is needed rather than import the whole module (which contains more useless functions which python will waste time importing them)
    - Whether you import a module or import a function from a module, Python will parse the whole module. Either way the module is imported. "Importing a function" is nothing more than binding the function to a name. In fact import module is less work for interpreter than from module import func.

4a)Importance of docstring in testing the code semi-automatically using doctest.

Python has a module called *doctest* that allows us to run the tests that we include in docstrings all at once. It reports on whether the function calls return what we expect. When a failure occurs, we need to review our code to identify the problem.
We should also check the expected return value listed in the docstring to make sure that the expected value matches both the type contract and the description of the function.
Python has a useful approach to code documentation called the **docstring**. This is simply a block of quoted text summarizing the purpose and usage of a Python object. By convention, they are enclosed in triple double-quotes ("""..."""). For example:

```
def add(x, y):
   """Return the sum of x and y."""
   return x + y
```

The docstring is the first string literal to appear in an object's definition. Here, the object is the **add** function. An object's docstring shows up when you use the built-in "help" function:

```
>>> help(add)
Help on function add in module __main__:

add(x, y)
   Return the sum of x and y.
```

You can look up any object's docstring via the __doc__ attribute:

```
>>> print add.__doc__
```

```
Return the sum of x and y.
```

A module may also have a docstring; again, it's simply the first block of quoted text to appear in the module:

```
#! /usr/bin/env python
# opts.py

"""This module provides an interface for defining
command-line-style options, and for reading and
storing their values as specified by the user.
"""

[module code]
```

The docstring for a module should be like a mini-manual for using the module. A descriptive opening sentence should tell users the purpose of the module, or a summary of what it does, followed by a mention of what classes and functions are most significant.

t

b)Python code to find the roots of a quadratic equation:

```
# import complex math module
import cmath

#a = 1
#b = 5
#c = 6

# To take coefficient input from the users
a = float(input('Enter a: '))
b = float(input('Enter b: '))
c = float(input('Enter c: '))

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)

print('The solution are {0} and {1}'.format(sol1,sol2))
```

a)l=[5,4,7,3,6,2,1]

i)Insert an element 9 at the beginning of the list

l.insert(0,9)

[9,5,4,7,3,6,2,1]

ii) Insert an element 8 at the end of the list

end=len(l)

l.insert(end,8)

[5,4,7,3,6,2,1,8]

iii) Insert an element 8 at the index position 3 of the list

l.insert(3,8)

[5,4,7,8,3,6,2,1]

iv)Delete an element at the beginning of the list

l.pop([0])

[4,7,3,6,2,1]

v) Delete an element at the end of the list

end=len(l)

l.pop([end])

[5,4,7,3,6,2]

vi) Delete an element at the index position 3

l.pop([3])

[5,4,7,6,2,1]

vii)**Print the list in reverse order**

```
def rev(l):
        r=[]
        for I in l:
              r.insert(0,i)
        return r
```

[1,2,6,3,7,4,5]

viii)Delete all elements of the list.

```
For I in len(l):
    l.pop([i])
```

empty list

b**)# Python program to check if the input number is prime or not**

```
num = 407

# take input from the user
# num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
  # check for factors
  for i in range(2,num):
     if (num % i) == 0:
```

```
        print(num,"is not a prime number")
        print(i,"times",num//i,"is",num)
        break
  else:
     print(num,"is a prime number")

# if input number is less than
# or equal to 1, it is not prime
else:
  print(num,"is not a prime number")
```

6 **a) Differences between a list and a string in Python:**

i) One simple difference between strings and lists is that lists can any type of data i.e. integers, characters, strings etc, while strings can only hold a set of characters.

ii) Though it sounds scary, mutation is actually a very simple concept. What Mutation means is that we can change the value of a list after we have created it, but we cannot in case of strings, one might question that we can change a value of a string by using operators such as concatenation, assignment etc

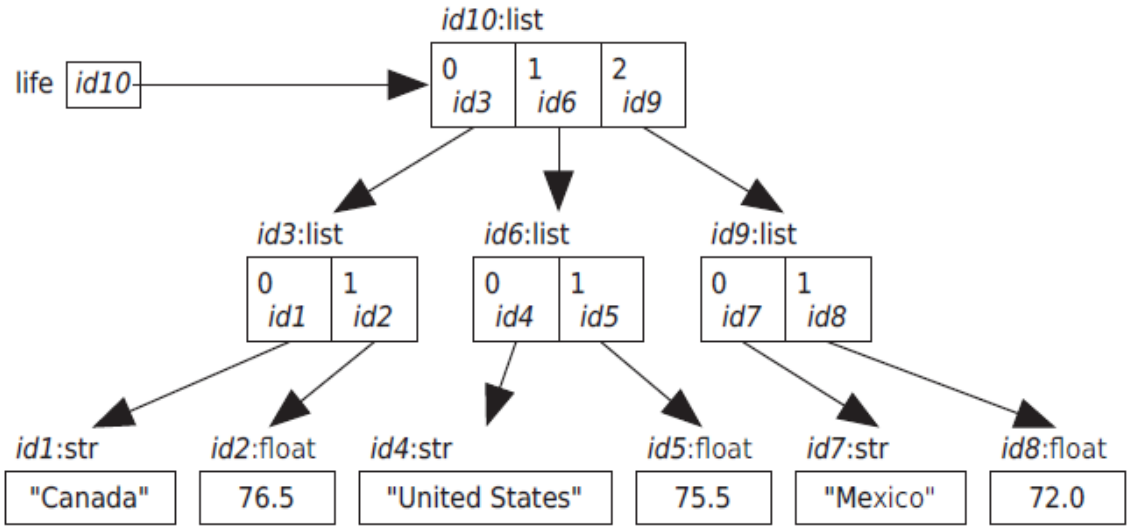
b) Python program to read a string with punctuations and print the same string without punctuations:

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

my_str = "Hello!!!, he said ---and went."

# To take input from the user
# my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
  if char not in punctuations:
     no_punct = no_punct + char

# display the unpunctuated string
print(no_punct)
```

c) What is a list of lists. Give an example along with its memory model.

A list whose items are lists is called a *nested list*. For example, the following nested list describes
life expectancies in different countries:

>>> life = [['Canada', 76.5], ['United States', 75.5], ['Mexico', 72.0]]
Here is the memory model that results from execution of that assignment



7a)

How can we use 'with' statement while opening a text file?

The with Statement

Because every call on function open should have a corresponding call on method close, Python provides a with statement that automatically closes a file when the end of the block is reached. Here is the same example using a with statement:

```
with open('file_example.txt', 'r') as file:
    contents = file.read()
```

The general form of a with statement is as follows:

```
with open(«filename», «mode») as «variable»:
    «block»
```

b) **Union,Difference ,Symmetric difference and Intersection of lows=[0,1,2,3,4] amd odds=[1,3,5,7,9]**

```
>>> lows = set([0, 1, 2, 3, 4])
>>> odds = set([1, 3, 5, 7, 9])
>>> lows - odds            # Equivalent to lows.difference(odds)
{0, 2, 4}
>>> lows & odds            # Equivalent to lows.intersection(odds)
{1, 3}
>>> lows <= odds           # Equivalent to lows.issubset(odds)
False
>>> lows >= odds           # Equivalent to lows.issuperset(odds)
False
>>> lows | odds            # Equivalent to lows.union(odds)
{0, 1, 2, 3, 4, 5, 7, 9}
>>> lows ^ odds            # Equivalent to lows.symmetric_difference(odds)
{0, 2, 4, 5, 7, 9}
```

c) Python program to read a word and print the number of letters,vowels and percentage of vowels in the word using a dictionary:

```
vowel_count = 0
count = 0
total = 0
space = 0
vowels='aeiou'
string = raw_input("Type a sentence and I will count the vowels!").lower()

for char in string:

   total += 1

   if char in 'aeiou':

      vowel_count += 1

   elif char ==' ':

      space += 1
   else:
      count += 1
print vowel_count
print space
print count
print (float)(vowel_count/count)*100

# make a dictionary with each vowel a key and value 0
count1 = {}.fromkeys(vowels,0)
```

```
# count the vowels
for char in string:
  if char in count1:
     count1[char] += 1

print(count1)
#print('vowel count is {0}.format(vowel_count)')
#print('letter count is {0}.format(count)')
```

8 a)

My_set={{‘india’,91},{‘USA’,1},{‘UK’,41},{‘Japan’,81}}
My_list=[[‘india’,91],[‘USA’,1],[‘UK’,41],[‘Japan’,81]]
My_dict={‘india’:91,‘USA’:1,‘UK’:41,‘Japan’:81}

b) **In what situations are the sets more useful than the lists.**

Sets are significantly faster when it comes to determining if an object is present in the set (as in x in s), but are slower than lists when it comes to iterating over their contents. When you want to store some values which you'll be iterating over, Python's list constructs are slightly faster. However, if you'll be storing (unique) values in order to check for their existence, then sets are significantly faster.

It turns out tuples perform in almost exactly the same way as lists, but they do use less memory by removing the ability to modify them after creation (immutable).

c) **Python program to read the content s of a text file and write into another**

```
with open("test.txt") as f:
  with open("out.txt", "w") as f1:
    for line in f:
      f1.write(line)
```

9 a) Write short notes on
i)instance

Function isinstance reports whether an object is an *instance* of a class—that is, whether an object has a particular type:

```
>>> isinstance('abc', str)
True
>>> isinstance(55.2, str)
False
```

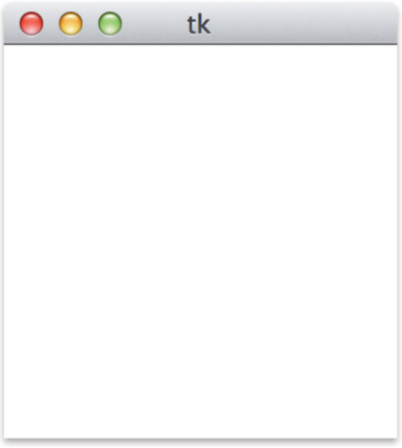
'abc' is an instance of str, but 55.2 is not.

ii**)__init__()**
Method __init__ is called whenever a class object is created. Its purpose is to initialize the new object; this method is sometimes called a *constructor*. Here are the steps that Python follows when creating an object:
1. It creates an object at a particular memory address.
2. It calls method __init__, passing in the new object into the parameter self.
3. It produces that object's memory address.

b) **Different components of a tkinter program:**
Here is a small but complete tkinter program:

```
import tkinter
window = tkinter.Tk()
window.mainloop()
```

Tk is a class that represents the *root window* of a tkinter GUI. This root window's mainloop method handles all the events for the GUI, so it's important to createonly one instance of Tk.
Here is the resulting GUI:



The call on method mainloop doesn't exit until the window is destroyed (which happens when you click the appropriate widget in the title bar of the window), so any code following that call won't be executed until later
c)**python program to create time objects current_time and bread_time and display the total time taken by the bread maker to prepare a bread**

```
import datetime

class Time:
   def __init__(self, hours, minutes, seconds):
      self.hours = hours
      self.minutes = minutes
      self.seconds = seconds


Time curtime,breadtime
```

```
ct = datetime.datetime.now().strftime('%H, %M, %S')
curtime('9,12,20')
print ct

Class MyTime:

   def __init__(self, hrs=0, mins=0, secs=0):
      """ Create a MyTime object initialized to hrs, mins, secs """
      self.hours = hrs
      self.minutes = mins
      self.seconds = secs

   def __str__(self):
      timeString = ""
      if self.hours < 10:
         timeString += "0"
      timeString += str(self.hours) + ":"
      if self.minutes < 10:
         timeString += "0"
      timeString += str(self.minutes) + ":"
      if self.seconds < 10:
         timeString += "0"
      timeString += str(self.seconds)
      return timeString

def add_time(t1, t2):
   h = t1.hours + t2.hours
   m = t1.minutes + t2.minutes
   s = t1.seconds + t2.seconds
   sumTime = MyTime(h, m, s)
   return sumTime

currentTime = MyTime(9, 14, 30)
breadTime = MyTime(3, 35, 0)
doneTime = add_time(currentTime, breadTime)
print(doneTime)
```

10 a)

Steps that python follows while creating an object:

Method __init__ is called whenever a Book object is created. Its purpose is to initialize the new object; this method is sometimes called a *constructor*. Here are the steps that Python follows when creating an object:

1. It creates an object at a particular memory address.
2. It calls method __init__, passing in the new object into the parameter self.
3. It produces that object's memory address.

b)**MVC Design with the help of tkinter program :**

Models, on the other hand, store data, like a piece of text or the current inclination of a telescope. They also don't do calculations; their job is simply to keep track of the application's current state (and, in some cases, to save that state to a file or database and reload it later).Controllers are the pieces that convert user input into calls on functions in the model that manipulate the data. The controller is what decides whether two gene sequences match well enough to be colored green or whether someone is allowed to overwrite an old results file. Controllers may update an application's models, which in turn can trigger changes to its views.

```
import tkinter
def click():
  counter.set(counter.get()+1)
if __name__=='__main__':
    window=tkinter.Tk()
    window.geometry("170x200+30+30")
    #The model
    counter=tkinter.IntVar()
    counter.set(0)
    #view model
    frame=tkinter.Frame(window)
    frame.pack()
    button=tkinter.Button(frame,text='Click',command=click)
    button.pack()
    label=tkinter.Label(frame,textvariable=counter)
    label.pack()
    window.mainloop()
```

c) **Write a Tkinter program to design a GUI window that has a label of background color green and foreground color white:**

```
import tkinter
window=tkinter.Tk()

window.geometry("170x200+30+30")
button=tkinter.Label(window,text='Hello',bg='green',fg='white')
button.pack()
window.mainloop()
```