ACADEMIC YEAR 2017-2018 Hall Ticket No:

SRI VASAVI
INSTITUTE OF ENGINEERING & TECHNOLOGY
NANDAMURU. PEDANA - 521 369.

II B.Tech I SEM (R16) I Mid Examinations **Subject: Python Programming**
Branch: CSE **Time: 9:15 AM To 10.45 AM**
Max. Marks : 30M **Date:16-08-2017**

Answer All Questions. All Questions carry equal marks.
3 X 10 = 30 Marks

1) a) Describe the Features of Python. **4 M**
b) Write a program to read and print values of variables of int, float, string and complex data types. **2 M**
c) What are the rules for identifiers in Python? **2 M**
d) What is Indentation? Explain importance of indentation in python. **2 M**

2) a) Explain range() Function with example. **3 M**
b) Write a Program to find the cube root of a given number. For example if the input is 8 output should be displayed 2. **3 M**
c) Write a Program to illustrate the continue, break and pass statements. **4 M**

3) a) What is Dictionary? What are the properties of dictionary keys. Explain fromkeys(), has_key(), get() methods of dictionary with an example. **6 M**
b) Write a Python Program for
i) Create and Assign Lists **ii)** Access Values in Lists
iii) Update Lists **iv)** Remove List Elements and Lists **4 M**

SRI VASAVI INSTITUTE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name of the Course: Python Programming | Academic Year: 2017 – 18
Name of the Faculty: Mr. P.V.L. Narasimha Rao | Year & Semester: II Year I Sem
Course Code: C214 | Section: CSE

Python Programming Mid – I Scheme of Valuation

1. A) Describe the features of python. - 4 Marks

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

1. B) Write a Program to read and print values of variables of int, float, string, and complex data types. - 2Marks

```
n1=int(input("Enter a integer value:"))
n2=float(input("Enter a Floating Point Value:"))
str=input("Enter a String:")
n3=complex(input("Enter a Complex Number:"))
print(n1)
print(n2)
print(str)
print(n3)
```

1. c) What are the rules for identifers in Python? - 2 Marks

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.
Here are naming conventions for Python identifiers −

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

1. d) What is Indentation? What is the use of Indentation? Explain with example. – 2 Marks

Whitespace at the beginning of the line is called indentation. These whitespaces or the indentation are very important in python. In a python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line. In python, indentation is used to associate the group statements.

Program to exhibit indentation error

```
[root@localhost ~]# vi try.py
Age=21
        Print("You can vote")

[root@localhost ~]# python3 try.py
  File "try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

The level of indentation groups statements to form a block of statements. This means that statements in a block must have the same indentation level. Python very strictly checks the indentation level and gives an error if indentation is not correct.

Like other programming languages python does not use curly braces to indicate blocks of code for class, function definitions or flow of control statements. It uses only indentation to form a block. All statements inside a block should be at the same indentation level.

The above example is corrected as below to get output

```
[root@localhost ~]# vi try.py
Age=21
Print("You can vote")

[root@localhost ~]# python3 try.py
You can vote
```

2. a) Explain range() function with example - 3 Marks

The range() type returns an immutable sequence of numbers between the given start integer to the stop integer.

range() constructor has two forms of definition:

range(stop)
range(start, stop[, step])

range() Parameters

range() takes mainly three arguments having the same use in both definitions:

- start - integer starting from which the sequence of integers is to be returned
- stop - integer before which the sequence of integers is to be returned.
  The range of integers end at stop - 1.
- step (Optional) - integer value which determines the increment between each integer in the sequence

Return value from range()

range() returns an immutable sequence object of numbers depending upon the definitions used:

range(stop)

- Returns a sequence of numbers starting from 0 to stop - 1
- Returns an empty sequence if stop is negative or 0.

range(start, stop[, step])

The return value is calculated by the following formula with the given constraints:

r[n] = start + step*n (for both positive and negative step)

where, n >=0 and r[n] < stop (for positive step)

where, n >= 0 and r[n] > stop (for negative step)

- (If no step) Step defaults to 1. Returns a sequence of numbers starting from start and ending at stop - 1.
- (if step is zero) Raises a ValueError exception

- (if step is non-zero) Checks if the value constraint is met and returns a sequence according to the formula
  If it doesn't meet the value constraint, Empty sequence is returned.

Example

```
# empty range
print(list(range(0)))

# using range(stop)
print(list(range(10)))

# using range(start, stop)
print(list(range(1, 10)))
```

Output:

```
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2. b) Write a program to find the cube root of a given number. For example if the input is 8 output should be displayed 2. - 3 Marks

```
num=int(input("Enter a number:"))
cube=num**(1/3)
print("Cube of a given number=",cube)
```

2. c) Write a program to illustrate continue, break, and pass statements. – 4 Marks

```
for number in range(10):
        if number == 6:
                continue
        if number==9:
                break
        if number==3:
                pass
        print(number)
```

Output:

```
0
1
2
3
4
5
7
8
```

3. a) What is Dictionary? What are the properties of dictionary keys. Explain fromkeys(), has_key(), get() methods of dictionary with an example. - 5 Marks

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

```
#!/usr/bin/python
dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

fromkeys()

The method **fromkeys()** creates a new dictionary with keys from *seq* and *values* set to value.

Following is the syntax for **fromkeys()** method −

```
dict.fromkeys(seq[, value])
```

Parameters

- **seq** − This is the list of values which would be used for dictionary keys preparation.
- **value** − This is optional, if provided then value would be set to this value

Return Value

This method returns the list.

get()

The method **get()** returns a value for the given key. If key is not available then returns default value None.

Following is the syntax for **get()** method −

dict.get(key, default = None)

Parameters

- **key** − This is the Key to be searched in the dictionary.
- **default** − This is the Value to be returned in case key does not exist.

Return Value

This method return a value for the given key. If key is not available, then returns default value None.

has_key()

The method has_key() returns true if a given *key* is available in the dictionary, otherwise it returns a false.

Following is the syntax for has_key() method −

dict.has_key(key)

Parameters

- key − This is the Key to be searched in the dictionary.

Return Value

This method return true if a given key is available in the dictionary, otherwise it returns a false.

3. b) Write a Python program for i) Create and Assign Lists ii) Access Values in Lists iii) Update Lists iv) Remove List Elements and Lists. - 5 Marks

```
#creating Lists
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
print(list1)
print(list2)
#Accessing Values in List
print("list1[0]: ", list1[0])
print("list2[1:5]: ", list2[1:5])
#Updating Lists
print("Value available at index 2 in list1 : ")
print(list1[2])
list1[2] = 2001;
print("New value available at index 2 list1 : ")
print(list1[2])
#Removing List Elements
print("Before deleting value at index 2 : ")
print(list1)
del list1[2];
print("After deleting value at index 2 : ")
print(list1)
```

SRI VASAVI
INSTITUTE OF ENGINEERING & TECHNOLOGY
NANDAMURU. PEDANA - 521 369.

II B.Tech I SEM II Mid Examinations **Subject: Python Programming**
Branch: CSE **Time: 9:15 AM To 10.45 AM**
Max. Marks : 30M **Date: 14-10-2017**

Answer All Questions. All Questions carry equal marks.
3 X 10 = 30 Marks

1. **A**. Explain keyword arguments, variable length arguments with examples. **5 M**

B. Write a function is_leap_year which takes the year as its argument and checks whether the year is leap year or not and then displays an appropriate message on the screen. **5 M**

2. **A**. How do you implement inheritance in Python? Describe in detail about multiple inheritance in Python. **5 M**

B. Explain in detail about handling exceptions and raising exceptions. **5 M**

3. **A**. What is unit testing? Write a test case to check the function factorial which returns the factorial of a given number **5 M**

B. Program to illustrate multithreading in python. **5 M**

SRI VASAVI INSTITUTE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name of the Course: Python Programming	Academic Year: 2017 – 18
Name of the Faculty: Mr. P.V.L. Narasimha Rao	Year & Semester: II Year I Sem
Course Code: C214	Section: CSE

Python Programming Mid – II Scheme of Valuation

1. A) Explain keyword and variable-length arguments. 5 Marks

Keyword arguments:

We call a function with some values; these values get assigned to the arguments according to their position.

For example, in the above function greet(), when we called it as greet("Bruce","How do you do?"), the value "Bruce" gets assigned to the argument name and similarly "How do you do?" to msg.
Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.

```
>>> # 2 keyword arguments
>>> greet(name = "Bruce",msg = "How do you do?")

>>> # 2 keyword arguments (out of order)
>>> greet(msg = "How do you do?",name = "Bruce")

>>> # 1 positional, 1 keyword argument
>>> greet("Bruce",msg = "How do you do?")
```

As we can see, we can mix positional arguments with keyword arguments during a function call. But we must keep in mind that keyword arguments must follow positional arguments.
Having a positional argument after keyword arguments will result into errors. For example the function call as follows:

```
greet(name="Bruce","How do you do?")
```

Will result into error as:

```
SyntaxError: non-keyword arg after keyword arg
```

Variable-length arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length*arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this −

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
```

```
   function_suite
   return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example

```
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

When the above code is executed, it produces the following result −

```
Output is:
10
Output is:
70
60
50
```

1. B) Write a function is_leap_year which takes the year as its argument and checks whether the year is leap year or not and then displays an appropriate message on the screen - 5Marks

```
def leapyear(year):
        if (year % 4) == 0:
                if (year % 100) == 0:
                        if (year % 400) == 0:
                                print("{0} is a leap year".format(year))
                        else:
                                print("{0} is not a leap year".format(year))
                else:
                        print("{0} is a leap year".format(year))
        else:
                print("{0} is not a leap year".format(year))
year = int(input("Enter a year: "))
leapyear(year)
```

2. A) How do you implement inheritance in Python? Describe in detail about multiple inheritance in Python.. – 5 Marks

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

Python Inheritance Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

Multiple inheritance: When a child class inherits from multiple parent classes, it is called as multiple inheritance.

Example

```
class Base1(object):
    def __init__(self):
        self.str1 = "Geek1"
        print "Base1"

class Base2(object):
    def __init__(self):
        self.str2 = "Geek2"
        print "Base2"

class Derived(Base1, Base2):
    def __init__(self):

        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print "Derived"

    def printStrs(self):
        print(self.str1, self.str2)

ob = Derived()
ob.printStrs()
```

2. B) Explain in detail about handling exceptions and raising exceptions. - 5 Marks

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of *try....except...else* blocks −

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Here are few important points about the above-mentioned syntax −

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

Example

This example tries to open a file where you do not have write permission, so it raises an exception −

```
#!/usr/bin/python

try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can\'t find file or read data"
else:
   print "Written content in the file successfully"
```

Raising an Exceptions

We can raise exceptions in several ways by using the raise statement. The general syntax for the raise statement is as follows.

Syntax

raise [Exception [, args [, traceback]]]

Here, *Exception* is the type of exception (for example, NameError) and *argument* is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows −

```
def functionName( level ):
  if level < 1:
    raise "Invalid level!", level
    # The code below to this would not be executed
    # if we raise the exception
```

3. A) What is unit testing? Write a test case to check the function factorial which returns the factorial of a given number. - 5 Marks

Unit testing is nothing but testing individual units or functions of a program.

Goals of unit testing:

- To make it easy to write test cases all a test needs to do is to say that, for this input, the function should give that result.
- To make it easy to run test cases. Usually this is done by clicking a one buttion or by typing a one key stroke.
- To make it easy, to tell if the test is passed. The framework takes care of reporting results. It either passed, or it provides a detailed list of failures.

In this example we will write a file *factorial.py*.

```
import sys
def fact(n):
    """
    Factorial function
    :arg n: Number
    :returns: factorial of n

    """
    if n == 0:
```

```
        return 1
    return n * fact(n -1)

def div(n):
    """
    Just divide
    """
    res = 10 / n
    return res


def main(n):
    res = fact(n)
    print(res)

if __name__ == '__main__':
    if len(sys.argv) > 1:
        main(int(sys.argv[1]))
```

Now we will write our first test case.

```
import unittest
from factorial import fact

class TestFactorial(unittest.TestCase):
    """
    Our basic test class
    """
    def test_fact(self):
        """
        The actual test.
        Any method which starts with ``test_`` will considered as a test case.
        """
        res = fact(5)
        self.assertEqual(res, 120)

if __name__ == '__main__':
    unittest.main()
```

Running the test:

```
$ python factorial_test.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

3. B. Program to illustrate multithreading in python. - 5 Marks

```
import _thread
import time
def display(threadname,delay):
        count=0
        while(count<=5):
                #print("%s:%s"%(threadname,time.ctime(time.time())))
                count+=1
                time.sleep(delay)
                print("%s:%s"%(threadname,time.ctime(time.time())))
try:
        _thread.start_new_thread(display,("ONE",1))
        _thread.start_new_thread(display,("TWO",2))
except:
        print("Error")
```

SRI VASAVI
INSTITUTE OF ENGINEERING & TECHNOLOGY
NANDAMURU. PEDANA - 521 369.

III B.Tech II SEM I Mid Examinations **Subject: Web Technologies**
Branch: CSE **Time: 9:15 AM To 10.45 AM**
Max. Marks: 30M **Date: 24-01-2018**

Answer All Questions. All Questions carry equal marks.
3 X 10 = 30 Marks

1. **a)** Define frame. Create a HTML page that displays multiple frames in a window. 3 M
b) Create a HTML which uses CSS that gives all H1 and H2 elements a padding of 0.5 ems; a grooved border style and a margin of 0.5 ems. 3 M
c) Write a script that asks the user to enter two numbers, obtains the two numbers from the user and outputs text that displays the sum, product, difference and quotient of the two numbers. 4 M

2. **a)** Create a XML document to store voter ID, voter name, address and date of birth details. Create a DTD to validate the document. 6 M
b) Compare and contrast between DOM and SAX parsers. 4 M

3. **a)** Explain working mechanism of AJAX with suitable example. 6 M
b) What is WSDL? Explain its features and document structure. 4 M

SRI VASAVI INSTITUTE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Examination: III B.Tech II sem I mid **subject:WT**
Branch:CSE **Max.marks:30**

Scheme of valuation

1.a) Define frame. Create a HTML page that displays multiple frames in a window.

Sol: HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Following is the example to create three horizontal frames −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "/html/top_frame.htm" />
    <frame name = "main" src = "/html/main_frame.htm" />
    <frame name = "bottom" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>

  </frameset>

</html>
```

Following is the example to create three vertical frames −

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>
```

```
  <frameset cols = "25%,50%,25%">
    <frame name = "left" src = "/html/top_frame.htm" />
    <frame name = "center" src = "/html/main_frame.htm" />
    <frame name = "right" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>
  </frameset>

</html>
```

b) Create a HTML which uses CSS that gives all H1 and H2 elements a padding of 0.5 ems; a grooved border style and a margin of 0.5 ems.

Sol:

```
<html>
    <head>
        <style>

            h1, h2 {
                padding: .5ems;
                border-style:groove;
                margin: .5ems;
            }
        </style>
    </head>
    <body>

        <h1>Hello World!</h1>
        <h2>Smaller heading!</h2>

    </body>
</html>
```

c) Write a script that asks the user to enter two numbers, obtains the two numbers from the user and outputs text that displays the sum, product, difference and quotient of the two numbers.

Sol:

```
<html>
    <head>
        <title>Integer Application</title>
         <script>

            var firstNumber; // first string entered by user
            var secondNumber; // second string entered by user
            var number1; // first number to add
            var number2; // second number to add
```

```
                    var sum; // sum of number1 and number2
                    var product; //product of number1 and number2
                    var difference; //difference of number1 and number2
                    var quotient; //quotient of number1 and number2

                    // read in first number from user as a string
                    firstNumber = window.prompt( "Enter first integer" );

                    // read in second number from user as a string
                    secondNumber = window.prompt( "Enter second integer" );

                    // convert numbers from strings to integers
                    number1 = parseInt( firstNumber );
                    number2 = parseInt( secondNumber );

                    sum = number1 + number2; // add the numbers
                    product = number1 * number2; //multiply the integers
                    difference = number1 - number2; //subtract the integers
                    quotient = number1 / number2; //divide the integers

                    // display the results
                document.writeln( "<h1>The sum is " + sum + "</h1>" );
                document.writeIn( "<h1>The product is " + product + "</h1>");
                document.writeIn( "<h1>The difference is " + difference + "</h1>");
                document.writeIn( "<h1>The quotient is " + quotient + "</h1>");
            </script>
            </head><body></body>
        </html>
```

2. a) Create a XML document to store voter ID, voter name, address and date of birth details. Create a DTD to validate the document.

Sol:

```
<? xml version = "1.0" encoding = "UTF-8">

<! DOCTYPE Voter_Information

[ <! Element Voter_information (Id, Name, Address, Date_of_birth)>

  <! Element Id (#PCDATA)>

  <! Element Name (#PCDATA)>

  <! Element Address (#PCDATA)>

  <! Element Date_of_birth (#PCDATA)>

] >
```

<Voter_Information>

<Id> FGK99567 </Id>

<Name> Mohan Kumar </Name>

<Address> Assam </Address>

<Date_of_birth> 05-03-1991 </Date_of_birth>

</Voter_Information>

b) Compare and contrast between DOM and SAX parsers
sol:

THE SAX APPROACH

- The Simple API for XML (SAX) approach to processing is called event processing.
- The processor scans the XML document from beginning to end.
- Every time a syntactic structure of the document is recognized, the processor signals an event to the application by calling an event handler for the particular structure that was found.
- The syntactic structures of interest naturally include opening tags, attributes, text, and closing tags.
- The interfaces that describe the event handlers form the SAX API.

THE DOM APPROACH

- The Document Object Model (DOM) is an application programming interface (API) for HTML and XML
- It defines the logical structure of documents and the way a document is accessed and documents. manipulated
- Properties of DOM

o Programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

o Provides a standard programming interface that can be used in a wide variety of environments and applications.

o structural isomorphism.

- The DOM representation of an XML document has several advantages over the sequential listing provided by SAX
- First, it has an obvious advantage if any part of the document must be accessed more parsers. than once by the
- Second, if the application must perform any rearrangement of the elements of the application.

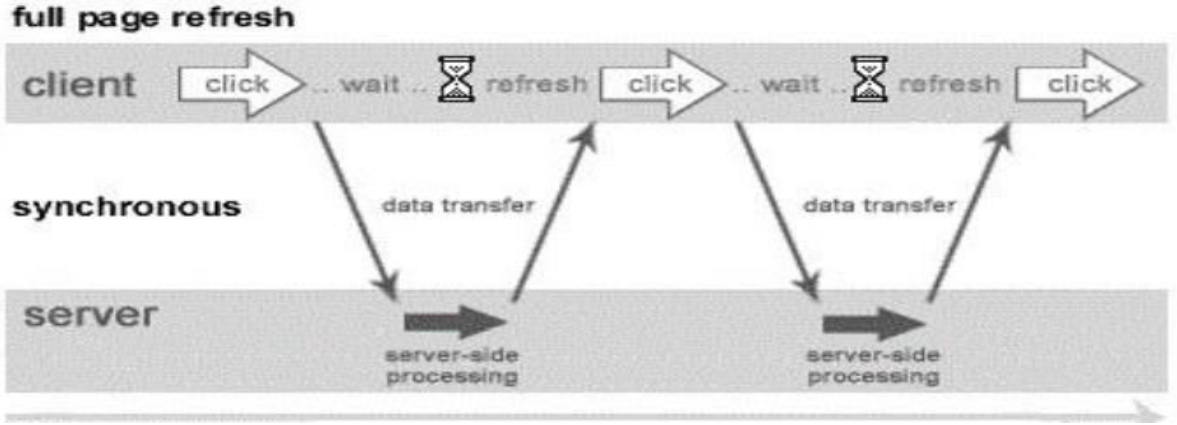
3. a) Explain working mechanism of AJAX with suitable example

Sol:
AJAX is an acronym for "**Asynchronous JavaScript And XML**". AJAX is a new technique for **creating** better, faster and **interactive web applications with the help of**

javascript, XML, DOM and HTML. AJAX allows you to send and retrieve data asynchronously **with out reloading the entire web page.**

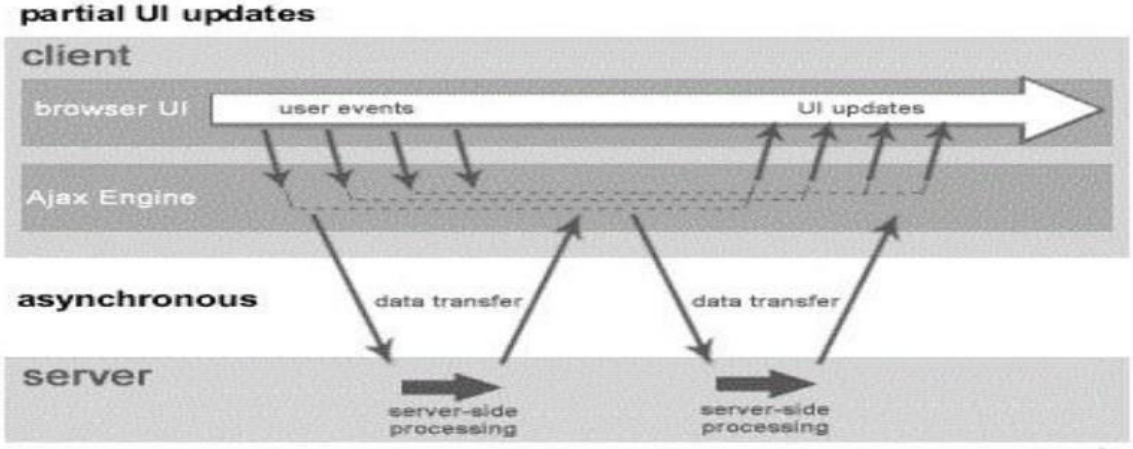
Synchronous vs Asynchronous mode of communication:

In traditional web applications, the synchronous mode of communication existed between the client and server is shown in the following figure.



In synchronous model, client will send the request to the server. Server will receive the request and then server will process the request. **At the time of processing the request the client must be in waiting mode. Client has to wait for the response** and meanwhile cannot perform any other operations(for ex: cannot send any other request to the server)

In AJAX -based web applications, the asynchronous mode of communication between the client and server is shown in the following figure.



In asynchronous model, client will send the request. Ajax engine will accept the request. AJAX engine will process the portion of the request(like data validations). The remaining **portion of the request which must be processed by the server will be send to the server by using the javascript and XMLHttpRequest object**. server will process the remaining portion of the request. Once AJAX engine is completed the processing of the request, it will send the response directly to the client. The engine;s interaction with the server does not interrupt the client's interaction with the application.

Example

```
<html>
<head>
<title>First AJAX Application</title>
```

```
<script language = "javascript">
        var xhr = false;
        // creating the XMLHttpRequest object
        if (window.XMLHttpRequest)
        {
                xhr = new XMLHttpRequest();
        }
else if (window.ActiveXObject)
        {
                xhr = new ActiveXObject("Microsoft.XMLHTTP");
        }
        function getData(dataSource, divID)
        {
                if(xhr)
                 {
                        var obj = document.getElementById(divID);
                        xhr.open("GET", dataSource); //setting  the request
                        xhr.onreadystatechange = function()          //callback function
                        {
                                if (xhr.readyState == 4 && xhr.status == 200)
                                 {
                                        obj.innerHTML =xhr.responseText;
                                 }
                        }
                        xhr.send(null);          //sending the request
                }
        }
</script>
</head>
<body>
      <h1>First Application using AJAX</h1>
     <img src="image1.jpg" onmouseover = “getData('bangles.txt','targetDiv')">
      <img src="image2.jpg" onmouseover = "getData('rings.txt','targetDiv')">
   <img src="image3.jpg" onmouseover = getData('necklaces.txt','targetDiv')">

<div id="targetDiv">
                <h1>Welcome to my Jewelery Showroom!</h1>
        </div>
</body>
</html>
```

In the above, web page is created that would display different jewellery items along with the some information. When the user points to an image whose information he wants to know, the details will be displayed on the web page with out the page being refreshed repeatedly

b) What is WSDL? Explain its features and document structure.

Sol;

WSDL stands for Web Services Description Language.

WSDL is the XML-based service representation language used to describe the details of the complete interfaces exposed by Web services and thus is the means to accessing a Web service.

WSDL is platform and language independent and is used primarily to describe SOAP enabled services.

Essentially, WSDL is used to describe precisely

– ***what*** a service does, i.e., the operations the service provides,

– ***where*** it resides, i.e., details of the protocol specific address, e.g., a URL, and

– ***how*** to invoke it, i.e., details of the data formats and protocols necessary to access the service's operations.

WSDL Elements:

A WSDL document contains the following elements:

definitions:

It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.

types:

Types are data type definitions used in message communication with the service.

message:

Message defines the messages exchanged with the service. Corresponding to each webservice message, the message section contains a message element which further contains sub elements.

portType:

It defines the web service operations to be performed, and messages to be invoked

The request-response type is the most common operation type, but WSDL defines four types:

Type	**Definition**
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

binding:

Binding specifies the protocols used to access each portType. A portType can have many bindings. In other words, a service might provide access to its methods(or operations) via numerous different protocols such as SOAP, HTTP etc.

service:

Service describes the set of ports to use when invoking a service. A service can have many ports, with each port having a name and a protocol binding.

The structure of WSDL document is as follows:

```
<definitions>

<types>
    datatypedefinitions.....
</types>

<message>
     definition of the data being communicated....
</message>

<portType>
    set of operations.....
</portType>

<binding>
     protocol and data format specification....
</binding>

</definitions>
```

SRI VASAVI
INSTITUTE OF ENGINEERING & TECHNOLOGY
NANDAMURU. PEDANA - 521 369.

III B.Tech II SEM II Mid Examinations **Subject: Web Technologies**
Branch: CSE **Time: 9:15 AM To 10.45 AM**
Max. Marks: 30M **Date: 23-03-2018**

Answer All Questions. All Questions carry equal marks.
3 X 10 = 30 Marks

1. **a)** Explain various PHP sort functions to sort array elements with an example. 5 M
b) How we can retrieve data in result set of MYSQL using PHP? Explain. 5 M

2. **a)** List and explain operators used in PERL. 5 M
b) What types of primary data structures are supported in PERL? Discuss. 5 M

3. **a)** Define class? How to create a class and its objects in ruby. How to declare and use constructor in ruby? 5 M
b) Explain the looping structures available in Ruby. 5 M

SRI VASAVI INSTITUTE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Examination: III B.Tech II sem II mid **subject:WT**
Branch:CSE **Max.marks:30**

Scheme of valuation

1.(a)

Sorting functions: The various sorting functions for arrays in php are
1. Sort()
2. rsort()
3. asort()
4. arsort()
5. ksort()
6. krsort()

sort(): The sort() function sorts an indexed array in ascending order.
Syntax: **sort(array,sortingtype);** Where array specifies the array to be sorted and sorting type specifies the how to compare the array elements which is optional.

rsort(): The rsort() function sorts an indexed array in descending order.
Syntax: **rsort(array, sortingtype);**

asort(): arsort() function sorts the associative array according to its value in ascending oreder.
Syntax: **asort(array, sortingtype);**

arsort(): arsort function sorts the associative array in descending order according to its value
syntax: **arsort(array,sortingtype);**

ksort(): ksort function sorts the associative array in ascending order depending on the key.
Syntax: **ksort(array,sortingtype);**

krsort(): krsort function sorts the associative array in descending order based on the key.
Syntax: krsort(array,sortingtype);

```
<html>
        <head>
                <title>sort function</title>
        </head>
        <body>
        <?php
        $a=array(22,45,63,75,1);
        echo "before sorting\n";
        print_r($a);
        sort($a);
        echo "after sorting\n";
        print_r($a);
        ?>
        </body>
</html>
```

1.(b) **Fetching the Records :**

a) mysql_fetch_row (Query handler) : Returns a record set from the database server, as a numerical array and moves the query handler to next record. if record does not exist return false.

b) mysql_fetch_assoc (Query handler) : Returns the record Set as an associative array with field names as its index position - if record does not exist return false..

c)

d) mysql_fetch_array (Query handler) : Returns the record set as an array with numerical or associative or both arrays . Returns false if no record found.

e) mysql_fetch_object (Query handler) : Returns the record set as an object with field names as its values .

Ex:

```
<?php
 mysql_connect("localhost", "root") or die(mysql_error());
echo "Connection to the server was successful!<br/>";
$database="regdb";
mysql_select_db($database) or die("unable to select database");
 $query="select * from regd";
mysql_query($query) or die(mysql_error());
$res=mysql_query($query);
?>
<table border="1" cellpadding="3">
 <tr>
 <td> name</td>
 <td>password</td>
<td>email</td>
<td>phone</td>
 <td>gender</td>
 <td>language</td>
<td>addresss</td>
</tr>
<?php
while($row=mysql_fetch_assoc($res))
 { ?>
<tr>
<td>
<?php echo $row['name'];?></td>
 <td><?php echo $row['pswd'];?></td>
<td><?php echo $row['email'];?></td>
 <td><?php echo $row['phone'];?></td>
<td><?php echo $row['gender'];?></td>
<td><?php echo $row['lang'];?></td>
<td><?php echo $row['address'];?></td>
</tr>
<?php }
?>
</table>
 <?php
mysql_close();
?>
```

2.(a) Perl language supports many operator types but following is a list of most frequently used operators:

- **Arithmetic Operators**
- **Relational Operators**
- **Logical Operators**
- **Assignment Operators**
- **Bitwise Operators**
- **Miscellaneous Operators**

Perl Arithmetic Operators Assume variable $a holds 10 and variable $b holds 20 then: **Operator**	**Description**	**Example**
+	Addition - Adds values on either side of the operator	$a + $b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	$a - $b will give -10
*	Multiplication - Multiplies values on either side of the operator	$a * $b will give 200
/	Division - Divides left hand operand by right hand operand	$b / $a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	$b % $a will give 0
**	Exponent – Performs exponential (power) calculation on operators	$a**$b will give 10 to the power 20

Perl Relational Operators Assume variable $a holds 10 and variable $b holds 20 **Operator**	**Description**	**Example**
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	($a == $b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	($a != $b) is true.
<=>	Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.	($a <=> $b) returns -1.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	($a > $b) is not true
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	($a < $b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	($a >= $b) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	($a <= $b) is true.

Perl Assignment Operators Assume variable $a holds 10 and variable $b holds 20 then: **Operator**	**Description**	**Example**
=	Simple assignment operator, Assigns values from right side operands to left side operand	$c = $a + $b will assign value of $a + $b into $c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$c += $a is equivalent to $c =$c + $a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$c -= $a is equivalent to $c = $c - $a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$c *= $a is equivalent to $c = $c * $a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$c /= $a is equivalent to $c = $c / $a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$c %= $a is equivalent to $c = $c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	$c **= $a is equivalent to $c = $c ** $a

2.(b)

Primary data structures supported in perl:

1. Scalar
2. Array (list), and
3. Association array (hash)

Scalars:

A Scalar variable always holds one value at a time and can take different kinds of values like numbers, strings and references.
A scalar variable has a *'$'* prefix. For example, $var = 'a string'; ## a quoted string $x = 12; $y=7.2; $z=0.7e2;
We can declare several scalars at once as follows. ($x,$y,$z)=(12,7.2,0.7e2);
Characters enclosed in single quotes are taken literally while variables are meaningful inside double quotes.

Arrays (Lists):

Array is a ordered list of scalars. In perl, array variables are preceded by an „@" sign.
Ex: @even=(2,4,6,8,10);
@color=('red','blue','yellow');
Arrays are Indexed by numbers with the first element of the array being indexed by zero. **When accessing an individual element of an array, write dollar sign with the variable name followed by index of an element in square brackets.**

```
@color=(„red“,“blue“,“yellow“);
print “first element of the array is $color[0]\n”;
 print “second element of the array is $color[1]\n”;
print “third element of the array is $color[2]\n”;
```

Hashes:

Perl associative arrays are called hashes. So a hash is a set of key/value pairs. Hash variables are preceded by the % prefix. You can create an hash arrays with the notation: (key1 => value1, key2 => value2, ...) **To access the single element of a hash, write the dollar sign with the hash variable name followed by key associated with the element in curly braces.**

```
Ex: %people = ( "pavan" => 7, "kumar” => 11 ); ## creating a hash
 print "$people{'pavan'}\n"; ## displays 7
print "$people{'kumar'}\n"; ## displays 11
```

3.(a)

Classes

A class is defined as a template for objects, of which any number can be created. An object has a state, which is maintained in its collection of instance variables, and a behavior, which is defined by its methods. An object can also have constants and a constructor.

The methods and variables of a class are defined in the syntactic container that has the following form:

class class_name
...
end

- Class names, like constant names, must begin with uppercase letters

Ex;

```
class Stack_2
def initialize(len = 5) ## Constructor
@stack = Array.new(len)
@max_len = len
@top = -1
end
def push(num) ##push method if @top == @max_len
puts "Stack is full"
else
end end
@top += 1
@stack[@top] = num
def pop() ##pop method if @top == -1
puts "Stack is empty"
else
end end
@top -= 1
def disp() ## printing method for i in 0..@top
print "#{@stack[i]} "
end
end
end
puts
my_stack = Stack_2.new(2)
my_stack.pop() my_stack.push(10)
my_stack.push(20)
my_stack.push(30)
my_stack.disp()
my_stack.push(40)
my_stack.pop()
 my_stack.pop()
 my_stack.disp()
```

3.(b) Loops in Ruby are used to execute the same block of code a specified number of times.

The while Statement:
Executes *code* while *condition* is true. A *while* loop's *condition* is separated from *code* by the reserved word *do*.

Syntax:
```
while condition [do]
code
 end
```

while modifier :

Syntax

```
begin
        code
end while condition
```

Executes *code* while *condition* is true. Irrespective of the condition, *code* is executed once before condition is evaluated.

Until Statement

Syntax:

```
until condition [do]
            code
end
```

Executes *code* while *condition* is false. An *until* statement's condition is separated from *code* by the reserved word *do*

until modifier:

Syntax:

```
begin
            code
 end until condition
```

Executes *code* while *condition* is false. If an *until* modifier follows a *begin* statement with no clauses, *code* is executed once before *conditional* is evaluated.