BBDNIIT

2025 / 2024-25 (ODD Semester)

Branch: CSE
Year/Section: Second Year (CS21, CS22, CS23, CS 24)

Name of Subject: Python Programming (BCC-302)

MODEL PAPER

WITH SOLUTION

Name of Faculty

Mr. Rehan Ahmad
Assistant Professor
COMPUTER SCIENCE & ENGINEERING

BABU BANARASI DAS
NORTHERN INDIA INSTITUTE OF TECHNOLOGY
LUCKNOW

BABU BANARASI DAS
NORTHERN INDIA INSTITUTE OF TECHNOLOGY, LUCKNOW
MODEL PAPER
ACADEMIC SESSION 2024-25(ODD SEMESTER)

Python Programming (BCC-302)

Time: 3 hrs **Max Marks: 70**

NOTE: Attempt all sections

SECTION A

Q01. Attempt ALL questions in brief: [7x02 =14]

	Question	Marks
1A	Can else be used with for loop in python? if so, when is it executed?	2
1B	Differentiate local and global variables?	2
1C	Describe about different functions of matplotlib and pandas.	2
1D	How is python interpreted language?	2

1E	Compute the output of following python code:
	
	`def Compute(s):`
	`    for str in s.split():`
	`        s="&".join(str)`
	`     return s`
	`print(Compute("PYTHON"))`
		2

1F	What is the output of following code:
	
	`def func(x, y=[]):`
	`    y.append(x)`
	`    return y`
	`print(func(2))`
	`print(func(3, []))`
	`print(func(4))`
		2

1G	What is file handling in Python, and why is it important?	2

SECTION B

Q 02. Attempt any THREE of the following: [7x3 =21]

	Question	Marks
2A	Write a program to generate Fibonacci Series upto 12 terms.	7
2B	Explain the continue, break, and pass statements with suitable example codes.	7
2C	Write a program that counts the occurance of a character in a string. Do not use built-in count function.	7
2D	Write a code in python to remove all duplicates from given L1=[1,2,3,4,5,6,7,8,5,4,3,4,5] using while loop	7
2E	What is Python IDE? Explain in detail?	7

SECTION C

Q03. Attempt Any One of the Following: [01x7 =7]

	Question	Marks
3A	Explain higher order function with respect to lambda expression. Write a Python code to Count occurrences of an element in a list?	7
3B	Write a program in python that accept comma separated sequence of words as input and prints the words in comma separated sequence after sorting them alphabetically. For example if input is : without,hello,bag,world Then output is :bag,hello,without,world	7

Q 04. Attempt Any One of the Following: [01x7 =7]

	Question	Marks
4A	Explain the numpy. Write an example of 2D and 3D array with the help of numpy	7
4B	Construct a function Perfect_Square(number) that returns a number if it is a perfect square otherwise it returns -1. For example, Perfect_Square(1) returns 1 Perfect_Square(2) returns -1	7

Q 05. Attempt Any One of the Following: [01x7 =7]

	Question	Marks
5A	What is string? Demonstrate five different built-in functions used in the string. Write a code to check whether a string is a palindrome or not.	7
5B	Write a program in python to generate all the rotation of given String without string slicing. For example if string is PYTHON the output will be: PYTHON, YTHONP,THONPY,HONPYT,ONPYTH,NPYTHO	7

Q 06. Attempt Any One of the Following: [01x7 =7]

	Question	Marks
6A	How to write a data in CSV file, give its syntax. Write a Python program to create a file with "Student_CS_Second_Year_056.txt" name and write the Section CS 21, Section CS22, Section CS23, Section CS24 in this file and then read this file to print it.	7

6B	What is broadcasting in NumPy? Write a code in python by plotting a simple line plot styles in matplotlib for given list of students and their marks
	Student=['John','Danish','Ananya','Alice','Raj']
	Marks =[60,30,75,30,50]
	Give the labels of x and y axix and also give title .
		7

Q 07. Attempt Any One of the Following: [01x7 =7]

	Question	Marks
7A	Explain the concept of DataFrame in Pandas. Write a python program to create a DataFrame from a dictionary and print it.	7
7B	Write a Python program that implements a temperature converter using Tkinter, allowing users to convert Celsius to Fahrenheit.	7

B.TECH. SECOND YEAR THIRD SEMESTER

MODEL PAPER SOLUTION

Python Programming BCC 302

Section A

1.A

Yes, the else clause can be used with for loop in Python. While using with for loop the else block executes after the loop finishes running, but only if the loop was not terminated prematurely using a break statement.

For Example,

```
Num = [1, 2, 3, 4, 5]
for i Num:
        if i==10:
                print("Number 10 found!")
                break
else:
   print("Number 10 not found.")
```

The loop iterates through all numbers.
Since 10 is not in the list, the break is never triggered, and the else block is executed

1.B

Local variables ae those that are defined in a function and have a local scope where global variables are those which are not defined inside a function and have global scope, The key differences between local and global variables are :

Sr.No.	**Aspect**	**Local Variable**	**Global Variable**
1	**Defined**	Inside a function or block	Outside all functions or blocks
2	**Scope**	Limited to the function or block	Accessible throughout the program
3	**Modification**	Modifiable only within the function	Requires global keyword to modify in functions
4	**Lifetime**	Exists during function execution	Exists as long as the program runs

1C

Matplotlib is a powerful visualization library in Python used for creating static, animated, and interactive plots. Commonly used functions in matplotlib are:

Function	**Description**
plt.plot()	Creates a basic line plot.
plt.hist()	Creates a histogram.
plt.xlabel()	Sets the label for the x-axis.
plt.ylabel()	Sets the label for the y-axis.
plt.title()	Sets the title of the plot.
plt.show()	Displays the plot.

Pandas is a Python library used for data manipulation and analysis. It provides data structures like Series and DataFrame. Commonly Used Functions in Pandas:

Function	**Description**
pd.read_csv()	Reads a CSV file into a DataFrame.
pd.read_excel()	Reads an Excel file into a DataFrame.
df.describe()	Generates summary statistics.
df.shape	Returns the number of rows and columns.
df.sort_values()	Sorts the DataFrame based on a column.
df.groupby()	Groups data based on a column.

1.D

Python is an interpreted language because its code is executed line by line at runtime by an interpreter, rather than being compiled into machine code before execution**.** The Python interpreter reads and executes each line of code sequentially. If an error occurs in one line, the execution stops immediately (unlike compiled languages, which detect errors during compilation).

When we run Python script the following steps happen.

Step 1: Bytecode Compilation (.pyc file)

The Python **interpreter** first converts the source code into an intermediate form called **bytecode** (not machine code).

Step 2: Execution by Python Virtual Machine (PVM)

The bytecode is executed by the **Python Virtual Machine (PVM)**, which interprets the code line by line and converts it into machine

instructions.**1.E.**
P&Y&T&H&O&N

1.F

```
 [1]
[1, 2]
[3]
[1, 2, 4]
```

1.G

File handling in Python is working with files in order to store, retrieve, and manipulate data. It allows programs to interact with external files, including text files, CSV files, binary files etc. File handling is crucial for data storage, logging, and managing user input/output operations.
Python provides built-in functions such as **open(), read(), write(), and close()** to handle files efficiently. The **with statement** ensures proper resource management by automatically closing files. File handling is used in applications like

- Web development
- Machine learning and
- Automation scripts etc. making it a fundamental skill for Python programmers**.**

Section B

2.A

```
# Program to display the Fibonacci sequence up to nth term (Here in this question code it
is 12)
nterms =12
n1, n2 = 0, 1# first two terms
count = 0
# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms = = 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
while count < nterms:
   print(n1)
   n3 = n1 + n2
# update values
   n1 = n2
   n2 = n3
   count += 1
```

2.B

The **continue statement** is used to skip the current iteration of a loop. When the python interpreter encounters a continue statement, then the rest of the statements in the loop are skipped and control is unconditionally transferred to the loop-continuation portion of the enclosing loops. It is useful when you want to skip certain parts of the loop under specific conditions but still want to continue the loop. The continue statement cannot be used without an enclosing for or while loop.

Python code to demonstrate the continue statement.

```
# Print all numbers from 1 to 5, but skip number 3
for i in range(1, 6):
    if i == 3:
        continue  # Skip the rest of the loop when i is 3
    print(i)
```

Output:

```
1
2
4
5
```

The **break statement** is used to terminate the execution of the nearest enclosing loop in which it appears. When the python interpreter encounters a break statement, control passes to the statement that follows the loop in which the break statement appears. Again, like continue statement, break statement cannot be used without an enclosing for or while loop.

Python code to demonstrate the break statement

```
# Print numbers from 1 to 5, but stop when the number is 3
for i in range(1, 6):
    if i == 3:
        break  # Exit the loop when i is 3
    print(i)
```

Output:

```
1
2
```

The **pass statement** is a placeholder that does nothing. It is used when a statement is syntactically required, but you do not want to execute any code. It specifies a null operation or simply No Operation (NOP) statement.

Python code to demonstrate the pass statement

```
# Loop over numbers from 1 to 5 and do nothing when the number is 3
for i in range(1, 6):
   if i == 3:
      pass  # Do nothing for 3, continue with the next iteration
   else:
      print(i)
```

Output:
```
1
2
4
5
```

2.C

```
def Count_Char(s,c):
        count=0
        for i in s:
                if i==c:
                        count+=1
        return count
str=input(“Enter a string :”)
ch=input (“Enter the character to be searched :”)
count=Count_Char(str,ch)
print (“In”,str,ch, “occurs”,count, “times”)
```

Output:
```
Enter a string : Google
Enter the character to be searched : e
In Google e occurs 1times
```

2.D

```
L1 = [1, 2, 3, 4, 5, 6, 7, 8, 5, 4, 3, 4, 5]
Unique_List = []
i = 0
while i < len(L1):
   if L1[i] not in Unique_List:
      Unique_List.append(L1[i])  # Append only if not present
   i += 1  # Increment index
print("List after removing duplicates:", Unique_List)
```

Output:
```
List after removing duplicates: [1, 2, 3, 4, 5, 6, 7, 8]
```

2.E

A **Python IDE (Integrated Development Environment)** is a software application that provides comprehensive tools to write, test, debug, and execute Python programs efficiently. It integrates various features like a text editor, debugger, syntax highlighting, code auto-completion, and execution capabilities within a single interface

Python IDEs simplify coding by providing the following features:

a) **Code Editor** – Helps write and edit Python code with syntax highlighting.
b) **Debugger** – Identifies and fixes errors in the code.
c) **Code Auto-Completion** – Suggests functions, variables, and keywords while typing.
d) **Integrated Terminal** – Runs Python scripts directly.
e) **Project Management** – Organizes multiple files and modules in one workspace.

Some popularly used Python IDEs:

1. PyCharm (JetBrains)

Features:

- Intelligent code suggestions & debugging.
- Integrated version control (Git).
- Supports frameworks like Django and Flask.
- Free (Community Edition) and Paid (Professional Edition).
- Widely used by professional developers and for large projects.

2. Visual Studio Code (VS Code)

- Lightweight, fast, and customizable.
- Extensions for Python development.
- Built-in terminal and debugging support.
- Widely used by beginners & intermediate developers.

3. IDLE (Python's Default IDE)

- Comes pre-installed with Python.
- Simple interface for writing and running scripts.

- Widely used by beginners and for small projects.

4. Jupyter Notebook
Features:

- Interactive environment for data analysis.
- Supports visualization and Markdown.
- Widely used in **Data Science and Machine Learning**.

5. Spyder

Features:

- Built-in scientific libraries (**NumPy, Pandas, Matplotlib**).
- Variable explorer to check data.
- Best for Researchers and Data Analysts.

Thus, Python IDEs make development easier by offering essential tools for writing, debugging, and executing Python programs efficiently. Choosing the **right IDE** depends on your project needs, expertise level, and workflow preferences.

3.A.
A Higher-Order Function (HOF) is a function that does at least one of the following:

1. Takes another function as an argument
2. Returns a function as a result

Higher-order functions help in functional programming, making code more modular, concise, and readable.
A **lambda function** is an **anonymous, single-line function** created using the lambda keyword. It does **not need a name** like a normal function.

- It is mainly used for short, simple operations.
- It is often used with higher-order functions like **map(), filter(), and reduce()**.

map() with Lambda

map() applies a function to each item in an iterable (list, tuple, etc.) and returns a new iterable.
Example: Squaring a List Using map()

```
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))  # Lambda squares each element
print(squared)
```

output: [1, 4, 9, 16, 25]

Working

- The lambda function lambda x: x**2 squares each number.
- map() applies this lambda function to each element in numbers.
- list(map(...)) converts the result to a list.

filter() with Lambda

filter() filters elements in an iterable based on a condition (True/False).

Example: Filtering Even Numbers Using filter()

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))  # Keeps only even
numbers
print(even_numbers)

output: [2, 4, 6, 8]
```

Working

- lambda x: x % 2 == 0 returns True if x is even.
- filter() applies this condition and keeps only the elements that return True.
- list(filter(...)) converts the result to a list.

reduce() with Lambda (from functools module)

reduce() applies a function **cumulatively** to the elements of an iterable, reducing them to a single value.

Example: Finding the Product of All Elements Using reduce()

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)  # Multiplies all numbers
together
print(product)

output: 120
```

Working

- reduce() starts with the first two numbers: 1 * 2 = 2.

- Then, it multiplies the result with the next number: 2 * 3 = 6, then 6 * 4 = 24, and so on.
- Final result is 120.

Thus advantages of using higher-order functions with lambda are:

- **Concise Code** – Reduces the need for lengthy function definitions.
- **Improves Readability** – Especially when performing operations like mapping or filtering.
- **Encourages Functional Programming** – Helps in writing modular, reusable code.

Python Program to count occurrence of element in a list

```
lst= [1,2,3,5,6,1,2,9,10,12,45,2]
item = int(input("Enter the element who's occurrence you want to count : "))
print(item, "occurrence is :", lst.count(item))
```

3.B

```
words = input("Enter comma-separated words: ")  # Accept input from user
word_list = words.split(",")  # Convert the input string into a list
word_list = sorted(word.strip() for word in word_list)
# Strip any extra spaces and sort the words alphabetically
sorted_words = ", ".join(word_list)
 # Join the sorted words back into a comma-separated string
print("Sorted words:", sorted_words) # Print the result
```

Working
Above **Python program** that accepts a **comma-separated sequence of words**, sorts them **alphabetically**, and prints them in a **comma-separated format**:
Processing:

- The input string is **split** into ["without","hello","bag","world "]
- The list is **sorted** alphabetically: ["bag","hello","without","world"]
- The words are **joined** back with commas.

Output:

```
Sorted words: bag, hello, without, world
```

Explanation:

1. **input()** reads the comma-separated words from the user.
2. **.split(",")** breaks the input into a list.
3. **word.strip()** removes extra spaces from each word.
4. **sorted()** arranges words alphabetically.
5. **", ".join()** joins the sorted words into a comma-separated string.

4.A

NumPy (Numerical Python) is a powerful library in Python for numerical and scientific computing. It provides support for multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Key Features of NumPy:

- **Efficient multi-dimensional arrays**: NumPy arrays are more efficient than Python's built-in lists, especially when it comes to large data.
- **Vectorized operations**: NumPy allows for operations on entire arrays without the need for explicit loops.
- **Broadcasting**: NumPy can perform operations on arrays of different shapes in a way that makes the smaller array expand to fit the larger one.
- **Linear algebra and random number generation**: NumPy includes functions for linear algebra, statistical operations, Fourier transforms, and random number generation.

Example of 2D and 3D Arrays in NumPy

Here's an example of creating and manipulating **2D** and **3D** arrays using NumPy.

```
import numpy as np
# Creating a 2D array (3x3 matrix)
array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("2D Array:")
print(array_2d)
```

Output:

```
2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

- The np.array() function is used to create a 2D array with 3 rows and 3 columns.
- Each row is represented as a list of elements.

<u>For Accessing a specific element in the array</u>

```
print(array_2d[0, 1])  # Accesses the element in the first row, second column
–
Output: 2
```

For Accessing a specific row

print(array_2d[1])

 Output: [4 5 6]

3D Array in NumPy
A **3D array** can be visualized as a stack of 2D arrays.

Example:

```
# Creating a 3D array (2x3x3 matrix)
array_3d = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                     [[10, 11, 12], [13, 14, 15], [16, 17, 18]]])

print("3D Array:")
print(array_3d)
```

Output:

```
3D Array:
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[10 11 12]
  [13 14 15]
  [16 17 18]]]
```

- This 3D array consists of 2 "layers" (2D arrays), where each layer contains 3 rows and 3 columns.

For Accessing a specific element in the 3D array

print(array_3d[1, 2, 0]) # Output: 16 (second layer, third row, first column)

For Accessing an entire "layer" (2D slice)

print(array_3d[0]) # Output: [[1 2 3], [4 5 6], [7 8 9]]

Briefly

- **2D Arrays** in NumPy are like matrices, and you can access elements using row and column indices.
- **3D Arrays** extend this concept to three dimensions (like a stack of matrices).
- NumPy makes it easy to perform complex mathematical operations on large arrays efficiently.

4.B

```
def Perfect_Square(number):
   # Find the integer square root by using integer division
   root = int(number ** 0.5)

   # Check if the square of the root equals the number
   if root * root == number:
      print('1')
   else:
      print('-1')
```

5.A.

In Python **string** is a sequence of characters enclosed within **single quotes (')** or **double quotes (")**.Strings are immutable, meaning they cannot be changed after they are created. Strings are one of the most commonly used data types in Python for storing and manipulating text-based data.

Some Built-in String Functions in Python:

1. **upper()**: Converts all characters in the string to uppercase.

```
text = "hello"
print(text.upper())  # Output: "HELLO"
```

2. **lower()**: Converts all characters in the string to lowercase.

```
text = "HELLO"
print(text.lower())  # Output: "hello"
```

3. **replace(old, new)**: Replaces all occurrences of a substring (old) with another substring (new).

```
text = "Hello World"
print(text.replace("World", "Python"))  # Output: "Hello Python"
```

4. **split(delimiter)**: Splits the string into a list of substrings based on the

given delimiter.

```
text = "Hello World Python"
print(text.split())  # Output: ['Hello', 'World', 'Python']
```

5. **strip()**: Removes leading and trailing whitespace characters.

```
text = "  Hello  "
print(text.strip())  # Output: "Hello"
```

6. **endswith(suffix)**: Returns True if the string ends with the specified suffix.

```
text = "Hello"
print(text.endswith("lo"))  # Output: True
```

7. **find(substring)**: Returns the index of the first occurrence of the substring, or -1 if not found.

```
text = "Hello World"
print(text.find("World"))  # Output: 6
```

8. **join(iterable)**: Joins the elements of an iterable (e.g., list) into a single string, using the string as a separator.

```
words = ["Hello", "World", "Python"]
print(" ".join(words))  # Output: "Hello World Python"
```

Code to Check if a String is a Palindrome:

A **palindrome** is a word, phrase, or sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and case).

```
# Input string
string = "madam"
# Initialize two pointers, one at the start and one at the end
start = 0
end = len(string) - 1
# Check if the characters from both ends match
is_palindrome = True
while start < end:
    if string[start] != string[end]:
        is_palindrome = False
        break
    start += 1
    end -= 1
if is_palindrome:
    print(f'"{string}" is a palindrome.')
else:
    print(f'"{string}" is not a palindrome.')
```

5.B

```
def generate_rotations(s):
    rotations = []
    temp = list(s)  # Convert string to a list for easy manipulation
    n = len(s)

    for _ in range(n):
        rotations.append("".join(temp))  # Append the current rotation
        first_char = temp.pop(0)  # Remove the first character
        temp.append(first_char)  # Append it to the end
    return rotations
string = input("Enter a string: ")
rotations = generate_rotations(string)

# Print output as comma-separated values
print(", ".join(rotations))
```

Working:

1. Convert the input **string into a list** (temp) for easy manipulation.
2. Iterate n times (where n is the length of the string).
3. Each time:
    a. Append the current version of temp (converted back to a string).
    b. Remove the **first character** (pop(0)) and **append it to the end**.
4. Finally, print all rotations as **comma-separated values**.

6.A

```
# Define the filename
filename = "Student_CS_Second_Year_056.txt"
# Writing to the file
with open(filename, "w") as file:
    file.write("Section CS21\n")
    file.write("Section CS22\n")
    file.write("Section CS23\n")
    file.write("Section CS24\n")
# Reading from the file
with open(filename, "r") as file:
    content = file.read()
# Printing the content
print("File Content:")
print(content)
```

6.B.

Broadcasting in NumPy is a powerful mechanism that allows arrays of different shapes to be combined and perform arithmetic operations element-wise without explicit looping.
NumPy automatically stretches the smaller array along the larger one's dimensions to match its shape.
For example, if you add a 1D array with shape (3,) to a 2D array with shape (2, 3), NumPy broadcasts the 1D array across each row of the 2D array. Broadcasting improves performance by avoiding explicit loops and making code more concise, but it requires that dimensions be compatible for broadcasting.

Python Program

```
import matplotlib.pyplot as plt
Student=['John','Danish','Ananya','Alice','Raj']
Marks =[60,30,75,30,50]
plt.xlabel("Students")
plt.ylabel("Marks")
plt.title("CLASS RECORDS")
plt.plot(Student, Marks)
plt.show()
```

7.A.

A DataFrame in Pandas is a 2D labeled data structure, similar to a **table in SQL**, **spreadsheet**, or a **dictionary of lists**. It consists of:

- **Rows and Columns**
- **Labeled Indexes** (for rows and columns)

Features of a Pandas DataFrame:

a) Stores **heterogeneous data** (integers, floats, strings, etc.).
b) Allows **row and column indexing** for easy data manipulation.
c) Supports operations like **filtering, sorting, aggregation, and transformations**.

Python Code for **creating and printing a DataFrame** using Pandas.

```
import pandas as pd  # Importing pandas library
# Creating a dictionary with data
data = {
   "Name": ["Rahul", "Arjun", "Ujjwal"],
   "Age": [20, 21, 22],
   "Grade": ["A", "B", "A"]
}
# Creating DataFrame from dictionary
df = pd.DataFrame(data)
# Printing the DataFrame
print("DataFrame:")
print(df)
```

Working:

1. Import Pandas (import pandas as pd)
2. Create a Dictionary (data)
   - Keys become column names.
   - Values are lists representing column values.
3. Convert Dictionary to DataFrame (pd.DataFrame(data))
4. Print DataFrame (print(df))

Output:

```
DataFrame:
    Name  Age Grade
0   Rahul   20     A
1    Arjun   21     B
2  Ujjwal   22     A
```

Each row has an index (0, 1, 2), and columns are labeled "Name", "Age", "Grade".

7.B

```
import tkinter as tk

# Function to convert from Celsius to Fahrenheit
def celsius_to_fahrenheit():
    celsius = float(celsius_entry.get())
    fahrenheit = (celsius * 9/5) + 32
    result_label.config(text=f"{celsius:.2f}°C = {fahrenheit:.2f}°F")
# Create the main window
parent = tk.Tk()
parent.title("Temperature Converter")

# Celsius to Fahrenheit Conversion
celsius_label = tk.Label(parent, text="Input Celsius:")
celsius_label.grid(row=0, column=0)

celsius_entry = tk.Entry(parent)
celsius_entry.grid(row=0, column=1)

c_to_f_button = tk.Button(parent, text="Convert to Fahrenheit",
command=celsius_to_fahrenheit)
c_to_f_button.grid(row=0, column=2)
# Display the result
result_label = tk.Label(parent, text="", font=("Helvetica", 14))
result_label.grid(row=2, columnspan=3)

# Start the Tkinter event loop
parent.mainloop()
```

Working:

In the codecabove -

- Temperature values can be entered in Celsius
- The result label will appear when they click the corresponding conversion button.
- As a result of the Tkinter event loop (root.mainloop()), the GUI is kept running and responsive to user input.