

## Python Unit-II, Assignment - II

### 1. Illustrate different types of decision statements used in Python programming.

**Decision statements** make a decision and change the flow of the program based on that decision. These are also called **Selection statements or Conditional statements**.

The decisions are made based on a given **condition** that results in a boolean result of either **True** or **False**.

There are 3 types of decision statements

Type	1. Single-Selection	2. Two-Way Selection	3. Multi-Way Selection
Command	if statement	if - else statement	a. Nested <b>if - else</b> statements b. <b>elif</b> Ladder statements

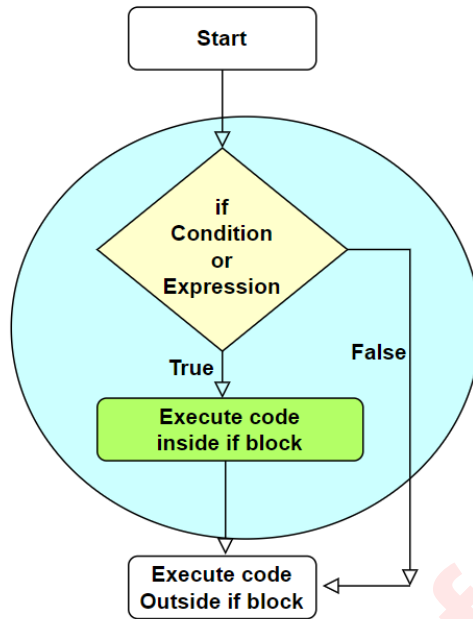
#### 1, Single Selection in Python (“if” statement only)

##### Definition:

- “if” is a simple selection statement in Python. It is used to modify the flow of execution of a program.
- “if” consists of a **condition (boolean expression)**, **colon :** and a **block of statements with the same indentation**,
  - When the condition is **True**, the **‘if’ block of statements** will be executed,
  - but when the condition is **False**, the flow comes out of “if” block & continues with next program statements
- All the statements inside the ‘if’ block must have the same indentation with spaces

##### Syntax:

```
if condition:
    True block of statements
Statements outside if-else block
```



**Application for “if”:**

**#if statement example**

```

m, n = 77, 87
if(m < n):
    result = "m is smaller than n"
    print(result)
    
```

**Output:**

m is smaller than n

**2. Two-Way Selection in Python (“if-else” statement)**

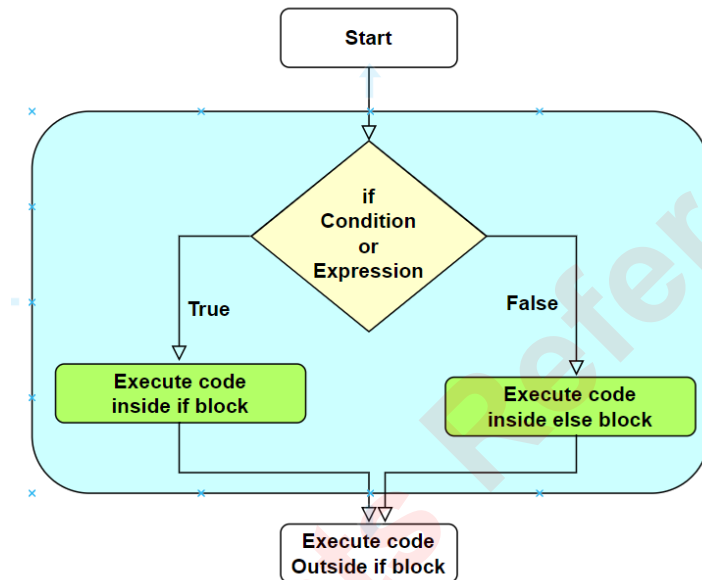
**Definition:**

- **if-else** is a two-way decision statement to decide between two alternative choices.
- **if** statement will have a condition; **else** statement has NO condition.
- **if-else** consists of
  - a condition (Boolean\_Expression),
  - a block of statements for ‘if’, and
  - A block of statements for ‘else’.
- When the condition is **True**, the ‘if’ block of statements will be executed
- When the Boolean\_Expression is **False**, the ‘else’ block of statements will be executed
- **Indentation:**
  - All the statements inside the ‘if’ block **must have the same indentation** with spaces
  - All the statements inside the ‘else’ block **must have the same indentation** with spaces
  - **However, a different indentation can be used for ‘if’ block and ‘else’ block.**

**Syntax:**

```

if condition:
    True block of statements
else:
    False block of statements
Statements outside if-else block
    
```



**Application for “if-else”:**

```

# Aim: Program to Check whether the given number is Even or Odd.
num = int(input("Enter an integer : "));
# true if num is perfectly divisible by 2
if(num % 2 == 0):
    print("{} is even.".format(num))
else:
    print("{} is odd.".format(num))
# Notice that if block has 4 space indentation and else block has 2 space indentation
    
```

**Output:**

```

Enter an integer : 7
7 is odd.
Enter an integer : 4
4 is even.
    
```

**3. Multi-Way Selection - Nested “if-else”**

When a series of decisions is required, the multi-way selection statements are used.

There are 2 types of multi-way selection statements

- A. Nested “if-else” statements
- B. “elif” Ladder statements

### A. Nested if-else statements

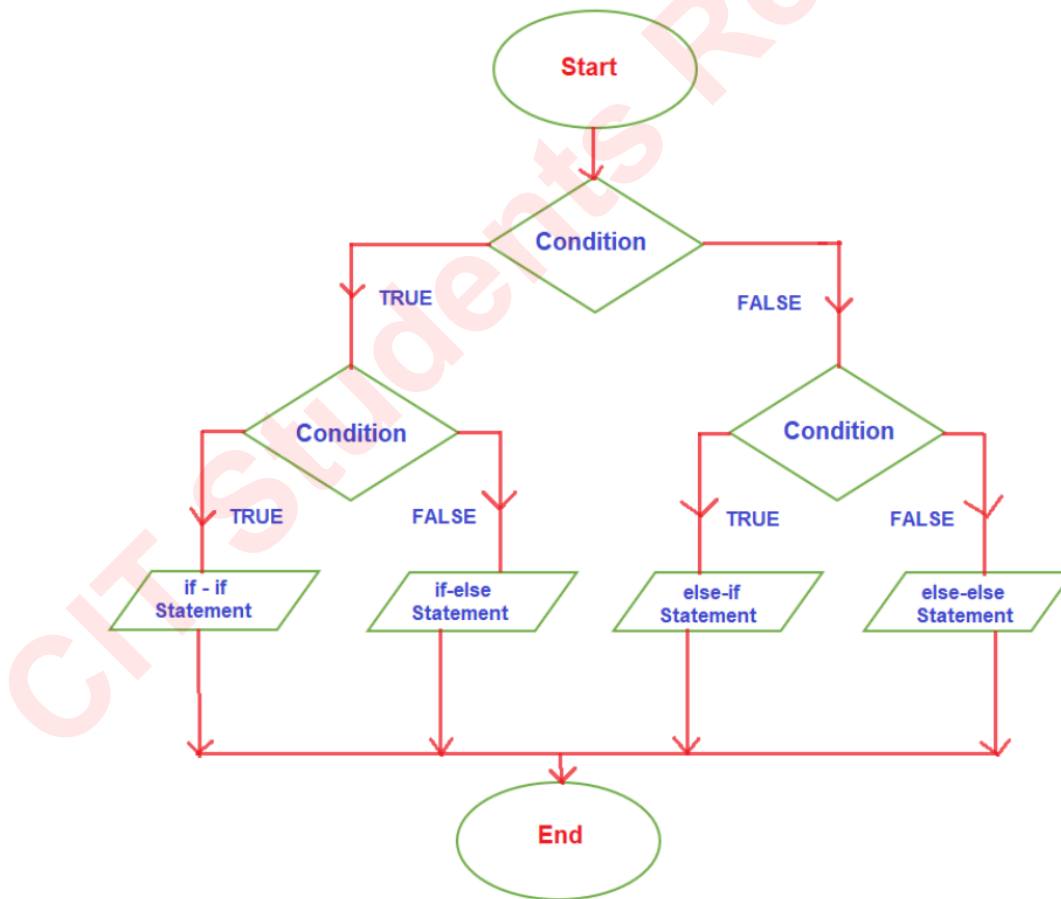
#### Definition:

Nesting means using one “if-else” construct within another “if-else” construct. Use nested “if-else” when you need to decide more within the parent “if” condition or parent “else” condition. **The nested “if-else” is used when multiple paths of decisions are required.**

#### Syntax:

```

if (Condition/Expression) :           # Outer if block
    if (Condition/Expression) :       # Inner if block
        Statements
    else:                             # Inner else block
        Statements
else:                                  # Outer else block
    if (Condition/Expression) :       # Inner if block
        Statements
    else:                             # Inner else block
        Statements
    
```



**Application for Nested “if-else”:**

```
#Checks whether input marks are pass or fail
# JustPass=40, pass>40, fail<40
marks = int(input("Enter marks 0-100 : "))
if marks >= 40:
    if marks==40:
        print("Just Passed with ", marks)
    else:
        print("Passed with ", marks);
else:
    print("Failed with ", marks)
```

**Output:**

Enter marks 0-100 : 75  
Passed with 75

Enter marks 0-100 : 40  
Just Passed with 40

Enter marks 0-100 : 30  
Failed with 30

**B. “elif” Ladder statements**

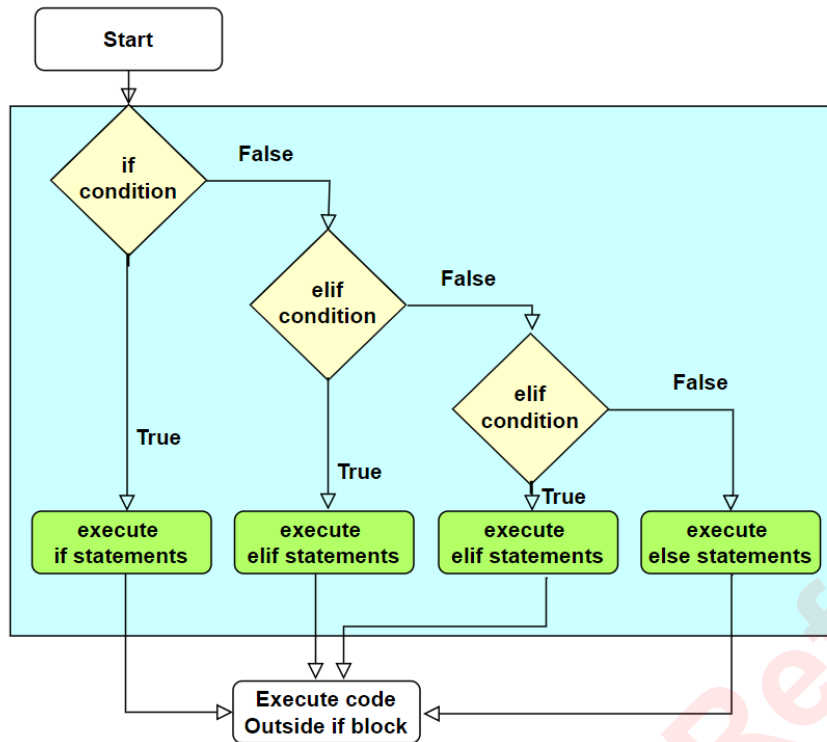
**Definition:**

In Python, “elif” keyword is a short form of “else if”. The “elif” is useful **when you need to decide a series of decisions** after each of the previous “if” conditions.

- The series of conditions are evaluated from **top to bottom**.
- When one condition becomes **true**, the statements of that condition will be executed and the control comes out of the whole “if” block.
- When all the conditions are **false**, then the last default “**else**” statement is executed and the control comes out of the whole “if” block.

**Syntax:**

```
if boolean_expression1:
    statement(s)
elif boolean_expression2:
    statement(s)
elif boolean_expression3:
    statement(s)
else:
    statement(s)
```



**Application-1:**

```

# elif conditional statements
x = 20
y = 70
if x > y:
    print("x is greater than y")
elif y > x:
    print ("y is greater than x")
else:
    print("x and y are equal")
    
```

**Output:**

y is greater than x

**2. a. Describe various repetition statements in Python with appropriate syntax. Explain how they are executed with the help of flow diagrams.**

In Python, **repetition statements** are used to execute a block of code repeatedly. Each repetition of a block of code is known as a **loop** or an **iteration**. So, the repetition statements are also called **looping or iterative structures**.

There are two types of repetition statements in Python..

1. **Indefinite or Condition Controlled Loop** - A loop that repeats a block of code until a condition is satisfied. (**while loop**)
2. **Finite or Sequence Controlled Loop** - A loop that repeats a block of code a specific number of times over a sequence of elements. (**for loop**)

Python provides two repetition or looping or iterative statements,

1. **'while' loop**
2. **'for' loop**
3. **Nested loops**

**1. The 'while' loop:**

**Definition:**

A **"while" loop** executes a block of statements repeatedly until the given condition is **True**.

The "while" loop is used when we DO NOT KNOW the number of iterations. It is also called an **Entry controlled or a Pre-Test loop**.

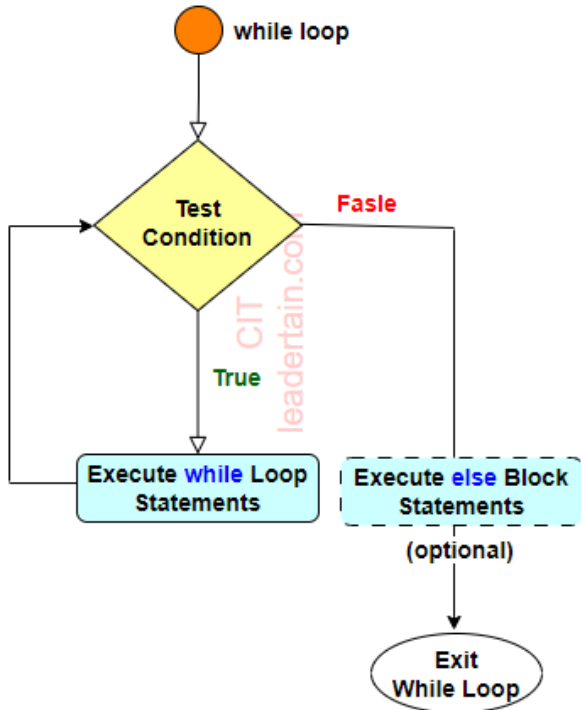
**Syntax of 'while' loop:**

```
Initialization (optional)
while (condition):
    Loop Body statements
    Incr or Decr (optional)
else: (optional)
    Block of statements
```

**Different ways to write 'while' loop condition:**

<b>while(true)</b> statements	<b>while( i&lt;5)</b> statements	<b>while( i&lt;=n)</b> statements
----------------------------------	-------------------------------------	--------------------------------------

**Flow Chart of 'while' loop:**



**WHILE loop:**

- “while” loop has a boolean **condition** that is checked before each iteration of the loop. If the condition is **True**, the code inside the while loop is executed. This will continue until the condition becomes **False**.

**WHILE loop with ELSE:**

- When the ‘while’ condition becomes **False**, the loop checks for the **optional** ‘else’ block.
  - If ‘else’ block is available, it executes the ‘else’ block and then exits the loop.
  - If ‘else’ block is not available, then simply exit the loop.

**2. The ‘for’ loop:**

**Definition:**

- A for loop is used to iterate over a sequence of elements such as string, range(), list, set, tuple or dictionary.
- The code inside the loop is executed repeatedly once for each element in the sequence.
- The “for” loop is used when we KNOW a number of iterations.



**Syntax: for**

```

for var in sequence:
    Loop body statements

else:    (optional)
    Block of statements
    
```

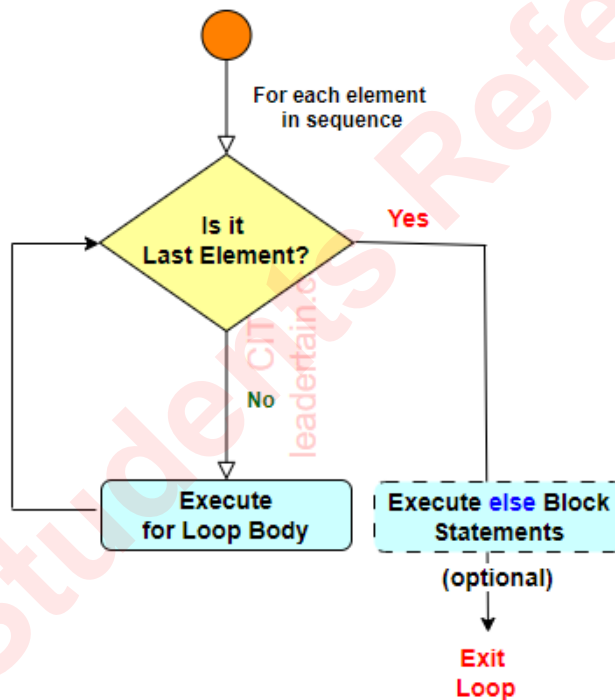
**var** - an iterator variable

**sequence** - a sequence of elements; a sequence can be a string, range(), list, set, tuple or dictionary

**Loop body statements** - a block of for loop statements

**else** - is an optional block in 'for'. When the for loop completes, it enters 'else' block of statements.

Flow chart - for loop



**Application:**

```

for loop using range() function: iterates through a range of values in sequence

# for loop using sequence of range() function
# range(start value, end value, Incr/decr)

print("Iterate in range(start, stop)")    #
for i in range(1,5): # 1-4, the 5 not included
    print(i)
    
```

**Output:**

Iterate in range(start, stop)

1  
2  
3  
4

### 3. Nested Loops

**Definition:**

Nested loop in Python is a loop that is placed inside another loop. The nested loops are used to iterate over multiple groups of data.

We can nest **for** and **while** loops in any way in Python. One such way is as follows:

- **for** loop nested with a **while** loop

**Purpose:**

Nested loops are typically used for working with patterns, and multidimensional data structures, such as printing two-dimensional arrays, iterating a list that contains a nested list.

**General Syntax: Nested Loop in Python**

**OuterLoop Expression:**

**InnerLoop Expression:**  
**Statements inside InnerLoop**

**Statements inside Outer\_Loop**

**Syntax: Nested “for - while” Loops**

**for** outer\_var in outer\_sequence:

**while** (condition):  
**Statements in inner while loop**

**Statements in outer for loop**

**Note:** Each iteration of the outer for loop triggers a complete iteration of the inner while loop

**Application: (Optional)**

**#Find Prime numbers in an Interval using nested “for - while” loop**

```
lownum = int(input("Enter low number of interval : "))  
highnum = int(input("Enter high number of interval : "))  
print("Prime numbers between", lownum, "and", highnum, "are:")
```

```

for num in range(lownum, highnum + 1):
    # Primes are always > 1
    if num > 1:
        i = 2
        while (i<num):
            if (num % i) == 0:
                break
            i += 1
        else:
            print(num, end=' ')

```

**Output:**

Enter low number of interval : 1  
Enter high number of interval : 20  
Prime numbers between 1 and 20 are:  
2 3 5 7 11 13 17 19

---

**b. Write a Python program to generate a Fibonacci series between a range, such as 0-n.****[Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, ...]**

# Aim: Generate Fibonacci series up to a given number of terms

n = int(input("Enter how many Fibonacci terms : "))

i = 0

# Term1 and Term2

term1, term2 = 0, 1

# Is the nth term positive?

if n &lt;= 0:

print("Enter a positive integer&gt;0.")

# If n is only 1 term

elif n == 1:

print("Fibonacci series of",n,"terms is:")

print(term1)

# Find and generate Fibonacci series up to n term

else:

print("Fibonacci series of",n,"terms: ")

while i &lt; n:

print(term1, end=" ")

next = term1 + term2

term1 = term2

term2 = next

i+=1

**Output:**

Enter how many Fibonacci terms: 5

Fibonacci series of 5 terms:

0 1 1 2 3

---

3. **a. Describe input validation loops and nested loops with appropriate examples.**

**Input validation loops:**

**Definition:**

An input validation loop prompts the user to enter input data, checks the input for validity, and repeats the prompt until valid input is entered. The loop continues until the user enters valid input and then the program can proceed with the remaining steps.

**Purpose:**

Input validation loops in Python ensure that the user enters valid input data. This is important because invalid data can cause errors or unexpected behavior in the program.

**Application:**

```
'''
Aim: Check the input number is valid. If invalid, then repeat the
prompt to reenter another number
'''
while True:
    user_input = input("Enter a number between 1 and 10 : ")
    num = int(user_input)
    if num < 1 or num > 10:
        print("Invalid number.")
    else:
        print("Valid number.")
        break
```

**Output:**

Enter a number between 1 and 10 : 27

Invalid number.

Enter a number between 1 and 10 : 7

Valid number.

**Explanation:**

In this example, the loop continues until the user enters a valid number between 1 and 10. The input is first converted to an integer using the int() function. If the input is an invalid integer, the loop continues. If the input is valid and within the range, the loop is exited and the program can proceed with the remaining steps.

**Nested loops:**  
(Refer to Q 2.a)

**b. Write a Python program to print a multiplication table of a given number.**

```
# Aim: Generate a multiplication table of a given number
t = int(input("Enter a number to generate multiplication table: "))
for j in range(1, 11):          # j ranges from 1 to 10
    print(t, "*", j, "=", t*j)
print()
```

**Output:**

3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30

**4. What is 'else' clause in looping structures in Python? Explain how it works with 'for' loop and 'while' loop with simple examples.**

**Definition:**

In Python, the "else" clause in looping structures is an optional block of code that is executed after the loop has completed all its iterations.

**Purpose:**

The purpose of the else clause is to provide a way to execute code when the loop has finished running normally, i.e., without being interrupted by a break statement.

**Syntax of "else" with loops:**

**for variable in iterable:**

# for loop statements

**else:**

# Statements after for loop has completed all iterations

**while condition:**

# while loop statements

**else:**

# statements after while loop has completed all iterations

- The “else” clause is executed only if the loop completes all its iterations without being interrupted by a break statement.
- In case, the loop gets terminated by the “break” statement, the else clause is not executed.

**Application: Using “for-else”:**

```
# Aim: Find a given number is prime or not
n = int(input("Enter a number: "))
for i in range(2, n):
    if n % i == 0:
        print(n, "is not a prime number")
        break
else:
    print(n, "is a prime number")
```

**Output:**

Enter a number: 7  
7 is a prime number

Enter a number: 6  
6 is not a prime number

**Application: Using “while-else”:**

```
# Aim: Find first 5 even numbers
count = 0
num = 0
while count < 5:
    num += 1
    if num % 2 == 0:
        print(num, end=' ')
        count += 1
else:
    print("\nAll 5 even numbers are printed")
```

**Output:**

2 4 6 8 10  
All 5 even numbers are printed

**5. a. Is string a mutable data type? Describe string slicing using the built-in slicing method and array slicing in detail with an example.**

**Strings are immutable**

In Python, **strings are immutable** data types. This means that once a string is created, it cannot be changed. If you want to modify a string, you need to create a new string with the desired changes.

Assume the given string is: **S = "cse"**

If you want to change the first character to 'C', you cannot simply do:

```
S[0] = "C" # raises an error because a string is immutable
```

Instead, you need to create a new string with the required change.

```
S = "C" + S[1:]    # Concatenates into a new string  
print(S)         # prints Cse
```

This creates a new string by concatenating the letter 'C' with the rest of the original string (i.e., s[1:]). The original string remains unchanged, and S now refers to the new string "Cse".

Explanation:

**String slicing**

**Strings** - A string in Python is an array of Unicode characters enclosed in quotes. The strings are indexed from 0 to n-1, where n is the size of the string. So characters in a string of size n can be accessed from 0 to n-1.

**String slicing** is the process of obtaining a range of characters or a substring of a string by using its indices. Following are the 2 methods to slice a string.

1. **Array slicing ( : operator)**
2. **slice() function**

1. **Array slicing ( : operator)**

**Definition:**

Array slicing is used to obtain a portion of a string array or a list. It uses the **slicing operator** : and square brackets to slice a string.

**Syntax:**

```
object [ start : stop : step ]
```

start - start index of the slice (included),

stop - end index of the slice (excluded), and

step - step size is the number of elements to skip between each element in the slice

**Application Array Slicing:**

```
s = "COLLEGE"
print(s[1:6])    # OLLEG    index 1 included, 6 excluded
print(s[1:6:2]) # OLG     index 1 included, 6 excluded
print(s[:3])    # COL     index 3 excluded
print(s[5:])    # GE      index 5 untill last index
# Negative index
print(s[-4:-1]) # LEG     index -4 included, -1 excluded
print(s[1:-4])  # OL     index 1 included, -4 excluded
print(s[5:1:-2]) # GL     index 5 included, 1 excluded, in Reverse order
# Reverse
print(s[::-1])  # EGELLOC String Reverse
```

**Optional** - This table shows how the string sequence is sliced using : operator

Index	0	1	2	3	4	5	6
s	C	O	L	L	E	G	E
s[1:6]	C	O	L	L	E	G	E
s[1:6:2]	C	O	L	L	E	G	E
s[:3] s[0:end]	C	O	L	L	E	G	E
s[5:] s[beg : ]	C	O	L	L	E	G	E

+ve index	0	1	2	3	4	5	6
-ve Index	-7	-6	-5	-4	-3	-2	-1
s[-4:-1]	C	O	L	L	E	G	E
s[1:-4]	C	O	L	L	E	G	E
s[5:1:-2]	C	O	L	L	E	G	E

Reverse a string							
s[::-1]	E	G	E	L	L	O	C



## 2. slice() Function

### **Definition:**

The slice() returns a slice object or a portion which is used to slice a sequence such as string, list, tuple, or range.

### **Syntax:**

```
slice ( start , stop , step )
```

start - start index of the slice (included),

stop - end index of the slice (excluded), and

step - step size is the number of elements to skip between each element in the slice

### **Application Array Slicing:**

```
# slice() function
s = "Our CIT College!"
sub = slice(0, 3)    # Creates a slice object representing [0:3]
result = s[sub]     # Slices the string s using the slice object sub
print(result)      # Output: "Our"
```

**Output:** Our

## **5. b. How do you encrypt and decrypt strings in Python? Explain them with suitable examples.**

### **Definition:**

The process of converting information that cannot be understood by the unauthorized user is called data encryption. The reverse process is called decryption. Data encryption is used to protect the information transmitted over the network. The encrypted data prevents data corruption, sniffing, stealing, or security attacks.

The network protocols such as FTPS and HTTPS do provide security to the information transmitted over the network.

### **Security attacks:**

Any action or a breach that compromises the security of information owned by an individual or an organization is called a security attack. Security attacks are classified into two: **Passive** and **Active**

- **Passive Attacks** - Unauthorized persons secretly reading or listening to private messages or message patterns while transmitting between a sender and a receiver.
- **Active Attacks** - Modification of the original data stream or the creation of a false data stream. Also includes,
  - Masquerade - one entity pretends to be a different entity
  - Replay- Passively capture and Unauthorized retransmission
  - DOS (Denial Of Service) - Disruption of an entire network

**Process of Data Encryption:**

- The information that is to be transmitted is called '**Plain Text**'.
- The **sender encrypts** the message by translating it into a secret code called '**Cipher Text**'.
- The **receiver decrypts** the cipher text into the original message or plain text.
- Both parties use **secret keys (public key & private key)** to encrypt and decrypt messages.
- **Caesar cipher** is a simple encryption method that has been in use for thousands of years.

**Caesar cipher Encryption:**

- Letter in a given plain text is changed to a letter that appears a certain number of positions farther down the alphabet set.
- For the characters near the end, the method goes back to the beginning of the alphabet set to locate the replacement characters.
- For example, if the distance value of a Caesar cipher is right-shift by 2 characters, the string "day" would be encrypted as "fca"

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b

**Application: Caesar cipher encryption**

```
# Caesar Cypher Encryption
msg = input('Enter your message: ')
dist= int(input('Enter cipher distance: '))
cmsg=""
for ch in msg:
    ordnum=ord(ch)
    ciphernum=ordnum+dist
    if ciphernum>ord('z'):
        ciphernum=ord('a')+dist-(ord('z')-ordnum+1)
    cmsg=cmsg+chr(ciphernum)
print(cmsg)
```

**Output:**

Enter your message: day  
 Enter cipher distance: 2  
 fca

### Application: Caesar cipher decryption

```
# Caesar Cypher Decryption
code=input('Enter your text: ')
dist=int(input('Enter distance: '))
msg=""
for ch in code:
    ordnum=ord(ch)
    ciphernum=ordnum-dist
    if ciphernum<ord('a') :
        ciphernum=ord('z') - (dist- (ord('a') -ordnum+1))
    msg=msg+chr(ciphernum)
print(msg)
```

#### Output:

```
Enter your text: fca
Enter distance: 2
day
```

**Tutorial Questions:**

**1. Define and Demonstrate the use of jump statements (break, continue, pass) in Python. Write a program to compute the sum of odd numbers within the given natural number using a continue statement.**

The Jump Statements are loop Control Statements in Python. The loop Control Statements are used to change the normal flow of a loop based on a condition.

The 3 jump or loop control statements in Python are,



1. **break** statement
2. **continue** statement
3. **pass** statement

**1. “break” statement**

**Definition:**

The “break” statement is used to exit a loop. It can be used in both while and for loops. The purpose of “break” statement is to end the execution of the loop (for or while) immediately and the program control goes to the statement after the loop. If there is an optional else statement in “for” or “while” loop, “break” also skips the optional “else” clause.

- The “break” statement almost always needs an “if” condition to work properly.
- The “break” statement is especially useful to quit from a nested loop (loop within a loop). It terminates the inner loop and the control shifts to the statement in the outer loop.

Syntax:	
<b>break</b>	
Using break in while loop:	Using break in for loop:
<pre style="font-family: monospace; color: #0056b3;">while (condition):     #statements     if (condition):         break     #statements</pre> 	<pre style="font-family: monospace; color: #0056b3;">for var in sequence:     #statements     if (condition):         break     #statements</pre> 

**Application:**

<p><b>BREAK in WHILE loop :</b> If the 'while' loop encounters an <b>optional</b> 'break', the loop simply <b>exits</b> even though the 'while' condition is <b>True</b>.</p>	<p><b>BREAK in FOR loop :</b> If the 'for' loop encounters an <b>optional</b> 'break', the loop simply <b>exits</b> even though the 'for' <b>sequence is incomplete</b>.</p>
<p><b>Application: Write a program to print 1-5 using a 'while' loop with 'break' to stop at 4.</b></p> <pre>i=1 # while loop with i = 1 to 3 while i &lt;= 5:     print(i)     i += 1     if(i==4):         break</pre> <p><b>Output:</b> 1 2 3</p> <p><b>Explanation:</b> The while loop continued until it encountered 4 and then exited while loop. The i += 1 statement increments the value of i by 1 on each iteration of the loop.</p>	<p><b>Application: Write a program to print 1-5 using a 'for' loop with 'break' to stop at 4.</b></p> <pre># for loop with i = 1 to 3 for i in range (1,6):     if(i==4):         break     print(i)</pre> <p><b>Output:</b> 1 2 3</p> <p><b>Explanation:</b> The 'for' loop continued until it encountered 4 and then exited 'for' loop.</p>



**2. "continue" statement**

**Definition:**

The "continue" statement forces the control to **skip** the current iteration, **not execute** the rest of the statements in the loop and **go to the next iteration** of the loop.

- A. In "while", the "continue" statement will directly **jump the execution control to "condition"**,
- B. In "for", the "continue" statement will **jump the execution control to the next element in the given sequence**.

<p><b>Syntax:</b></p>
<p><b>continue</b></p>

Using continue in while loop:	Using continue in for loop:
<pre> while (condition):     #statements     if (condition):         continue     #statements                     </pre> 	<pre> for var in sequence:     #statements     if (condition):         continue     #statements                     </pre> 

### Application:

<p><b>CONTINUE in WHILE loop:</b> If the 'while' loop encounters an <b>optional 'continue'</b>, the loop simply skips the current iteration and jumps to the 'condition' for next iteration.</p>	<p><b>CONTINUE in FOR loop :</b> If the loop encounters an <b>optional 'continue'</b>, the loop simply skips the current iteration and jumps to next iteration in the sequence.</p>
<p><b>Application: Program to print <u>even numbers between 1 and 5</u> using while loop &amp; continue (skip) on odds</b></p> <pre> n = 1 while n &lt; 5:     n += 1     if (n % 2) != 0:         continue     print(n)                     </pre> <p><b>Output:</b> 2 4</p> <p><b>Explanation:</b> The while loop continued until 2. When it encountered 3, the value incremented to 4 and executed 'continue' to skip the iteration.</p>	<p><b>Application: Program to print <u>even numbers between 1 and 5</u> using 'for' loop &amp; continue (skip) on odds</b></p> <pre> for n in range(1,6):     if (n % 2) != 0:         continue     print(n)                     </pre> <p><b>Output:</b> 2 4</p> <p><b>Explanation:</b> The 'for' loop continued until 2. When it encountered 3, it executed 'continue' to skip 3 and continued with 4 in the sequence.</p>

### 3. "pass" statement

#### Definition:

"pass" statement is a **placeholder for an empty code** in loops, functions, classes, or if statements. "pass" statement is a **null operation** and nothing happens when it is executed.

- Empty code causes errors in loops, functions, classes, or if statements.
- So, we can use "pass" statement to avoid errors.

**Syntax:** pass

**Application:**

<p><b>PASS in IF and WHILE loop</b></p> <pre>n=1 while (n&lt;5):     if (n==3):         pass     n += 1</pre>	<p><b>PASS in FOR loop</b></p> <pre>college = "Chalapathi" for i in college:     pass</pre>
<p><b>PASS in Function</b></p> <pre>def func():     pass func()</pre>	<p><b>PASS in Class</b></p> <pre>class name:     pass</pre>

**2. a. What are string format methods? Explain the string format operator % with examples.**

String formatting is the process of inserting a custom string or variable in predefined text.

Python allows string formatting using one of the following 4 methods.

1. % (String Format Operator)
2. format() method
3. f-strings
4. String Template Class (module: `from string import Template`)

**% (String Format Operator):**

The % Operator is called a String Format Operator or an Interpolation Operator. It is used for simple positional formatting in strings. It allows you to insert values into a string, replacing placeholders with actual values. The placeholders are represented by percent signs followed by a format specifier that defines the type of the value being inserted.

**Syntax:**

```
<"format specifiers"> % <data/vars>
```

- **format specifiers** - carries any string with %formatSpecifiers as placeholders (%d, %f, %s)
- ‘ % ’ is the **String Format Operator** that substitutes data/variable value into format specifier
- **data/vars** - values to replace format specifiers

<"format specifiers"> may have format specifiers with Padding for data values as specified below:

```
%<fieldwidth>.<precision>f    %6.2f
%<fieldwidth>d                %3d
%<fieldwidth>s                %10s
```

<fieldwidth> is the total number of digits given for the value  
 <precision> is the number of decimal digits out of the given total digits  
 The unfilled digit positions will be added as padding spaces on the left.

**Example:**

```
name = "Raj"
age = 25
marks = 75.55
# without padding
print("Name:%s, Age:%d, Marks:%f" % (name, age, marks))
```

**Output:** Name:Raj, Age:25, Marks:75.550000

```
# with padding
print("Name:%10s, Age:%3d, Marks:%6.2f" % (name, age, marks))
```

**Output:** Name: Raj, Age: 25, Marks: 75.55

**Table:** List of format specifiers in Python

Format Specifier	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E



**Table: Built-in string format methods**

Method	Description	s="software Engineers"
s.capitalize()	converts the first character to uppercase.	Software Engineers
s.upper()	Converts all the characters in a string to uppercase.	SOFTWARE ENGINEERS
s.lower()	Converts all the characters in a string to lowercase.	software engineers
s.isupper()	Returns True if all the characters are uppercase. Otherwise, False	False
s.islower()	Returns True if all the characters are lowercase. Or else False.	False
s.find(substring, [start, end])	Returns the index of a specified character in the string or the start position of the given substring.	s.find("Eng") 9
s.count(substring,[start,end])	Counts the occurrence of a character or substring in a string.	s.count("r") 2
s.expandtabs([tabsize])	Replaces tabs defined by \t with spaces. Default tabsize = 8	
s.endswith(substring, [start, end])	Returns True if a string ends with the specified substring. False otherwise.	s.endswith("neers") True
s.startswith(substring, [start, end])	Returns True if a string starts with the specified substring. False otherwise.	s.startswith("Soft") True
s.isalnum()	Return True if all characters in a string are alphanumeric. False otherwise.	False
s.isalpha()	Return True if all characters in a string are alphabetic. False otherwise.	True
s.isdigit()	Return True if all characters in a string are digits. False otherwise.	False
s.split([separator],[maxsplit])	Splits a string separated by a separator(defaults is whitespace) and an optional maxsplit to determine the split limit. Returns a list.	["Software","Engineers"]

s.join(iterable)	Joins all items in an iterable into a single string separated by the string s.	
s.replace(old, new,[maxreplace])	Replace old substring contained in the string s with a new substring.	s.(“Engineers”,”Programmer”) Software Programmers
s.swapcase()	Returns a new string with swapped case. i.e., uppercase becomes lowercase and vice versa.	sSOFTWARE pPROGRAMMERS
s.strip([characters])	Removes whitespaces or optional characters at the beginning and at the end of the string.	
s.lstrip([characters])	Removes leading whitespace or optional characters from a string.	
s.rstrip([characters])	Removes trailing spaces at the end of the string.	

**2. b. How + and \* operators work with strings?**

Python provides the following operators for string operations:

- **String concatenation operator “ + ”**
- **String repetition operator “ \* ”**
- String Slicing operator “ : ” to obtain substrings
- Indexing to traverse through strings,
- Membership operators (in, not in) to search for strings
- Relational operators (>, >=, <, <=) to compare strings

Here, we will discuss + and \* operators.

The + operator is used to concatenate 2 or more strings into one string.

The \* operator is used to repeat a string up to a given number of times.

Operator	Purpose	Operation	Description
+	Concatenation	s1 + s2	Concatenates two strings, s1 and s2.
*	Repetition	s * n	Makes n copies of string s.

**(+) Concatenation Operator:**

**Definition:**

The + operator is used to join or concatenate two strings.

This concatenation operator in Python concatenates only objects of the same type.

**Usage:**

```
concatenate_string = string1 + string2 # concatenate the two strings
```

### **(\*) Repetition Operator:**

#### **Definition:**

The \* operator is used to repeat a given string n number of times (similar to multiplication).

#### **Usage:**

```
repeat_string = string1 * n # repeats string1 n times
```

#### **Application:**

```
# Concatenate & Repetition of strings
```

```
s1 = "Computer "
```

```
s2 = "Science"
```

```
s3 = s1 + s2
```

```
print(s3)
```

```
s4 = s1*3
```

```
print(s4)
```

#### **Output:**

Computer Science

Computer Computer Computer

---

### **3. What are string padding functions in Python? Explain with simple examples.**

#### **Definition:**

In Python, String padding functions add extra characters such as spaces or zeros, at the start or end of a string to get a required length. Python does provide several built-in string padding functions for this purpose.

The commonly used string padding functions are,

1. **ljust()**,
2. **rjust()**, and
3. **center()**.

#### **Purpose:**

These methods are very useful for formatting text in the form of tables or displaying information in a fixed-width format.

1. **ljust()**

#### **Syntax:**

```
svar.ljust(width[, fillchar])
```

**ljust()** function returns left-justified string of given width. The string is padded with **fillchar** (default is space) to make up the length.

**Example:**

```
s = 'Guntur'
padded_s = s.ljust(10, '*')
print(padded_s)    # Guntur****
```

2. **rjust()****Syntax:**

```
svar.rjust(width[, fillchar])
```

**rjust()** function returns right-justified string of given width. The string is padded with **fillchar** (default is space) to make up the length.

**Example:**

```
s = 'Guntur'
padded_s = s.rjust(10, '*')
print(padded_s)    # ****Guntur
```

3. **center()****Syntax:**

```
svar.center(width[, fillchar])
```

**center()** function returns centered string in the given width. The string is padded with **fillchar** (default is space) to make up the length.

**Example:**

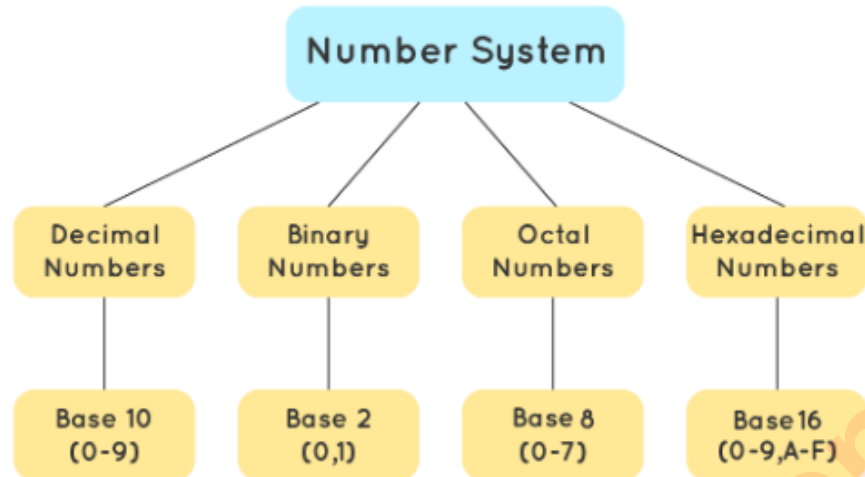
```
s = 'Guntur'
padded_s = s.center(10, '*')
print(padded_s)    # **Guntur**
```

#### **4. What are the different types of number systems? Write a program to convert a decimal number into binary and octal numbers.**

**Number systems** are the technique to represent numbers in the computer system architecture, every value that we save or read has a defined number system.

Computer architecture supports the following number systems.

1. **Binary number system**
2. **Octal number system**
3. **Decimal number system**
4. **Hexadecimal (hex) number system**



**1) Binary Number System (Base: 2, Digits: 0, 1)**

A Binary number system has only two digits 0 and 1. All binary numbers are represented in 0s and 1s.

**2) Octal number system (Base: 8, Digits: 0-7)**

Octal number system has only 8 digits from 0 to 7. All octal numbers are represented in 0,1,2,3,4,5,6 and 7.

**3) Decimal number system (Base: 10, Digits: 0-9)**

Decimal number system has only 10 digits from 0 to 9. All decimal numbers are represented in 0,1,2,3,4,5,6, 7,8, and 9.

**4) Hexadecimal number system (Base: 16, Digits: 0-9, A-F)**

A Hexadecimal number system has 16 alphanumeric values from 0 to 9 and A to F. All hexadecimal numbers are represented in 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E, and F. Here A is 10, B is 11, C is 12, D is 13, E is 14 and F is 15.

**Table: Number Systems & Representation in Python**

Number system	Base	Digits used	Example	Python assignment
Binary	2	0,1	(11110000) <sub>2</sub>	var = <b>0b</b> 11110000
Octal	8	0,1,2,3,4,5,6,7	(360) <sub>8</sub>	var = <b>0o</b> 360
Decimal	10	0,1,2,3,4,5,6,7,8,9	(240) <sub>10</sub>	var = 240
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	(F0) <sub>16</sub>	var = <b>0x</b> F0

**Decimal to Binary Conversion:**

- **Manual conversion** - Decimal number is divided by 2 until we get 1 or 0 as the final remainder.

$$28_{10} = 11100_2$$

Base target	Decimal	Remainder
2	28	0
2	14	0
2	7	1
2	3	1
2	1	

**Decimal to Octal Conversion:**

- **Manual conversion** - Decimal number is divided by 8 until we get 0 to 7 as the final remainder.

$$28_{10} = 34_8$$

Base target	Decimal	Remainder
8	28	4
8	3	

**Decimal to Hexadecimal Conversion:**

- **Manual conversion** - Decimal number is divided by 16 until we get 0 to 15 as the final remainder.

$$28_{10} = 1C_{16}$$

Base target	Decimal	Remainder
16	28	12 = C
16	1	

**Automatic conversion: Decimal to Binary, Octal, Hexadecimal**

From decimal to binary, octal or hexadecimal, use **bin( )**, **oct()**, **hex()** functions respectively.  
 From binary, octal or hexadecimal to decimal, use **int(other num, base )** function..

**Application:**

# Aim: Program to convert Decimal to Binary, Octal and Hexadecimal

# Decimal to Binary, Octal, Hexadecimal

```
n = 28
bn = bin(n)
ot = oct(n)
hx = hex(n)
print("Decimal to Binary ", n, "=", bn)
print("Decimal to Octal ", n, "=", ot)
print("Decimal to Hexadecimal ", n, "=", hx)
```

```
#Binary to Decimal
print("Binary to Decimal = ",int(bn,2))
#Octal to Decimal
print("Octal to Decimal = ",int(ot,8))
#Hexadecimal to Decimal
print("Hexa to Decimal = ",int(hx,16))
```

**Output:**

Decimal to Binary 28 = **0b**11100  
 Decimal to Octal 28 = **0o**34  
 Decimal to Hexadecimal 28 = **0x**1c  
 Binary to Decimal = 28  
 Octal to Decimal = 28  
 Hexa to Decimal = 28

**(OPTIONAL)**

**Binary to Decimal Conversion**

Binary Number = **11100**<sub>2</sub>

1	1	1	0	0
1x2 <sup>4</sup>	1x2 <sup>3</sup>	1x2 <sup>2</sup>	0x2 <sup>1</sup>	0x2 <sup>0</sup>
16	8	4	0	0

= 16 + 8 + 4 + 0 + 0  
**Decimal number = 28**<sub>10</sub>

### Octal to Decimal Conversion

Octal Number is :  $34_8$

3	4
$3 \times 8^1$	$4 \times 8^0$
24	4

$$= 24 + 4$$

$$\text{Decimal number} = 28_{10}$$

### Hexadecimal to Decimal Conversion

Hexadecimal Number is :  $1c_{16}$

1	c = 12
$1 \times 16^1$	$12 \times 16^0$
16	12

$$= 16 + 12 + 4$$

$$\text{Decimal number} = 28_{10}$$



List of Programs

1. Write a Python program to display all prime numbers up to 200.

```
No = int(input("Please Enter any No: "))
print("Prime Nos between", 1, "and", No, "are:")
for Number in range(1, No + 1):
    if Number > 1:
        for i in range(2, Number):
            if (Number % i) == 0:
                break
        else:
            print(Number)
```

**Output:**

```
Please Enter any No: 200
Prime Nos between 1 and 200 are:
2
3
5
7
...
```

2. Write a Python program to find the average of the top two test scores out of the three test scores received.

```
m1=int(input("Enter the marks1: "))
m2=int(input("Enter the marks2: "))
m3=int(input("Enter the marks3: "))
total=0
if (m1>m2) :
    if (m2>m3) :
        total=m1+m2
    else:
        total=m1+m3
elif (m1>m3) :
    total=m1+m2
else:
    total=m2+m3
avg=total/2
print("The avg of best two test marks is:",avg)
```

**Output:**

```
Enter the marks1: 80
Enter the marks2: 50
Enter the marks3: 90
The avg of best two test marks is: 85.0
```

**3. Write a Python program to check whether a given number is an Armstrong number**

```

n=int(input("Enter a number: "))
s = n
b = len(str(n))
sum1 = 0
while n != 0:
    r = n % 10
    sum1 = sum1+(r**b)
    n = n//10
if s == sum1:
    print("The given number", s, "is Armstrong number")
else:
    print("The given number", s, "is not Armstrong number")

```

**Output:**

Enter a number: 523

The given number 523 is not Armstrong number

Enter a number: 370

The given number 370 is Armstrong number

**4. Write a program to find whether the given string is a palindrome by using functions**

```

# Method-1 - Using Functions
def reverse(str1):
    if(len(str1) == 0):
        return str1
    else:
        return reverse(str1[1 : ]) + str1[0]

s1 = input("Enter a string : ")
s2 = reverse(s1)
print("String in reverse Order : ", s2)
if(s1 == s2):
    print("This string is a Palindrome")
else:
    print("This string is not a Palindrome")

# Method-2: Using if-else (Simple)
st = input("Enter a string : ")
if(st == st[::-1]):
    print("This string is a Palindrome")
else:
    print("This string is not a Palindrome")

```

**Output:**

Enter a string : GOOG  
This string is a Palindrome

Enter a string : CIT  
This string is not a Palindrome

**5. Write a Python program to sort given input strings.**

```
def sort_string(s):
    chars = list(s)
    n = len(chars)
    for i in range(n):
        for j in range(0, n-i-1):
            if chars[j] > chars[j+1]:
                chars[j], chars[j+1] = chars[j+1], chars[j]
    return ''.join(chars)

s = "Engineers"
print("Original string:", s)
print("String after sorting:", sort_string(s))
```

**Output:**

Original string: Engineers  
String after sorting: Eeeginnrs

**6. Write a Python program to perform string concatenation and copy operations.**

```
#Method 1 (Format())
var1 = "Guntur"
var2 = "City"
print("{} {}".format(var1, var2))
var3 = "{} {}".format(var1, var2)
print(var3)

# Method 2 (,comma)
var1 = "Guntur"
var2 = "City"
print(var1, var2)

# Method 3 (% operator)
var1 = "Guntur"
var2 = "City"
print("% s % s" % (var1, var2))

# Method 4 (+ operation)
var1 = "Guntur "
```

```

var2 = "City"
var3 = var1 + var2
print(var3)

# Method 5 (Join())
var1 = "Guntur"
var2 = "City"
print(" ".join([var1, var2]))
var3 = " ".join([var1, var2])
print(var3)

```

**Output:**

```

Guntur City
Guntur City
Guntur City
Guntur City
Guntur City
Guntur City
GunturCity
Guntur City

```

**7. Write a Python program to demonstrate traversal through a string using a loop.**

```

# Method 1:
print("Using for loop")
st1 = "High"
count= len(st1)
for i in range(count):
    print("At index",i,"=",st1[i])

# Method 2:
print("Using while loop")
st2 ="Score"
count= len(st2)
i= 0
while i < count :
    print("At index",i,"=",st2[i])
    i=i+1

```

**Output:**

```

Using for loop
At index 0 = H
At index 1 = i
At index 2 = g
At index 3 = h

```

### Using while loop

At index 0 = s  
At index 1 = c  
At index 2 = o  
At index 3 = r  
At index 4 = e

8. Write a Python program that interchanges the first and last characters of a given string.

```
txt = input('Enter a string: ')
newtxt = txt[-1]+txt[1:-1]+txt[0]
print('New string:', newtxt)
```

#### Output:

Enter a string: CSE  
New string: ESC

9. Write a Python program using 'while' loop to print the first N numbers divisible by 5

```
start = int(input("Enter start number:"))
end = int(input("Enter last number:"))
for i in range(start, end+1):
    if(i%5==0):
        print(i)
```

#### Output:

Enter start number:10  
Enter last number:30  
10  
15  
20  
25  
30

10. Write a Python program to calculate the number of seconds in a day.

```
def seconds_per_day(days):
    hours = days * 24
    minutes = hours * 60
    seconds = minutes * 60
    return seconds

print(seconds_per_day(2))
```

#### Output:

172800