UNIT -2

Control Statement: Definite iteration for Loop, Formatting Text for output, Selection if and if else Statement Conditional Iteration The While Loop

Strings and Text Files: Accessing Character and Substring in Strings, Data Encryption, Strings and Number Systems, String Methods Text Files.

CONTROL STATEMENTS :Looping Statements

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times.
- Looping Statements supported by Python :
  - **for**
  - **while**
- There are two types of loops—
  - those that repeat an action a predefined number of times**(definite iteration)**, and
  - those that perform the action until the program determines that it needs to stop **(indefinite iteration)**.

1 Definite iteration for Loop

- **Executing a Statement a Given Number of Times**
  - The form of this type of for loop is

```
for <variable> in range(<an integer expression>):
        <statement-1>
        .
        .
        <statement-n>
```

- The first line of code in a loop is sometimes called the **loop header,**which denotes the number of iterations that the loop performs.
  - The colon (:) ends the loop header.

Surya Lakshmi Kantham Vinti
CSE DEPT
ACET

- The **loop body comprises** the statements in the remaining lines of code, below the header.These statements are executed in sequence on each pass through the loop.
  - The statements in the loop body *must be indented and aligned in the same column.*

```
>>> for i in range(4):
        print(i)
0
1
2
3
```

Count-Controlled Loops:

- Loops that count through a range of numbers are also called **count-controlled loops.**
- **The** value of the count on each pass is often used in computations.
- To count from an explicit lower bound, the programmer can supply a second integer expression in the loop header. When two arguments are supplied to range, the count ranges from the first argument to the second argument minus 1
- The only thing in this version to be careful about is the second argument of range, which should specify an integer greater by 1 than the desired upper bound of the count.
- Here is the form of this version of the for loop:

  for <variable> in range(<lower bound>, <upper bound + 1>):

  <loop body>

```
>>>for i in range(5,10):
            print(i)
5
6
7
8
9
```

Loop Errors: Off-by-One Error:

- The loop fails to perform the expected number of iterations. Because this number is typically off by one, the error is called an **off-by-one error**

Traversing the Contents of a Data Sequence:

The values contained in any sequence can be visited by running a for loop , as follows:

for <variable> in <sequence>:

<do something with variable>

– On each pass through the loop, the variable is bound to or assigned the next value in the sequence, starting with the first one and ending with the last one.

```
>>> name="Surya Lakshmi"
>>> nl=[45,36,644]
>>> nt=(4,22,6,1)
>>> for i in name:              #Traversing string
        print(i,end=",")
S,u,r,y,a, ,L,a,k,s,h,m,i,
>>>for i in nl:                #Traversing list
        print(i,end=",")
45,36,644,
>>>for i in nt:                #Traversing tuple
        print(i,end=",")
4,22,6,1,
```

Specifying the Steps in the Range :

- A variant of Python's range function expects a third argument that allows you to nicely skip some numbers.
- The third argument specifies a step value, or the interval between the numbers used in the range, as shown in the examples that follow:

```
>>> list(range(1, 6, 1)) # Same as using two arguments
[1, 2, 3, 4, 5]
>>> list(range(1, 6, 2)) # Use every other number
```

[1, 3, 5]

>>> list(range(1, 6, 3)) # Use every third number

[1, 4]

Loops That Count Down

- Once in a while, a problem calls for counting in the opposite direction, from the upper bound down to the lower bound.
- a loop displays the count from 10 down to 1 to show how this would be done:

```
>>> for count in range(10, 0, -1):
        print(count, end = " ")
10  9 8 7 6 5 4 3 2 1
```

- When the step argument is a negative number, the range function generates a sequence of numbers from the first argument down to the second argument plus 1.

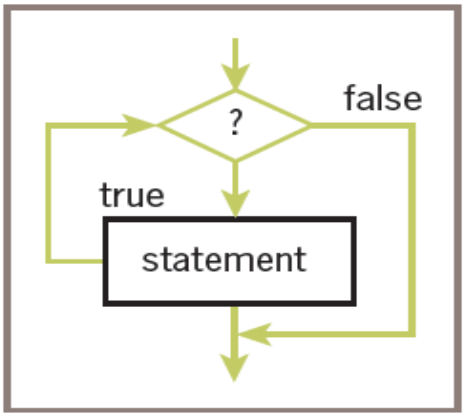
2. Conditional Iteration: The while Loop

- A loop continues to repeat as long as a condition remains true.This is called Conditional iteration
- **The Structure and Behavior of a while Loop :**
- Conditional iteration requires that a condition be tested within the loop to determine whether the loop should continue. Such a condition is called the loop's **continuation condition.**
  - If the continuation condition is false, the loop ends.
  - If the continuation condition is true, the statements within the loop are executed again.
- **Syntax:**

  while <condition>:

  <sequence of statements>

- The while loop is also called an **entry-control loop, because its condition is tested at** the top of the loop.
- This implies that the statements within the loop can execute zero or more times.




```
# Summation with a while loop

theSum = 0

count = 1

while  count <= 10:

theSum += count

count += 1

print(theSum)
```

- The while loop is also called an **entry-control loop, because its condition is tested at** the top of the loop.
  - This implies that the statements within the loop can execute zero or more times.

The while True Loop and the break Statement

- If the loop must run at least once, use a while True loop and delay the examination of the termination condition until it becomes available in the body of the loop.
- Ensure that something occurs in the loop to allow the condition to be checked and a break statement to be reached eventually.

```
while True:

    number = int(input("Enter the numeric grade: "))

    if number >= 0 and number <= 100:

            print(number) # Just echo the valid input

            break
```

else:

print("Error: grade must be between 100 and 0")

OUTPUT

A trial run with just this segment shows the following interaction:

Enter the numeric grade: 101

Error: grade must be between 100 and 0

Enter the numeric grade: –1

Error: grade must be between 100 and 0

Enter the numeric grade: 45

45

Random Numbers

- To simulate randomness in computer applications, programming languages include resources for generating **random numbers.**
- The function **random.randint (from module random)** returns a random number from among the numbers between the two arguments and including those numbers.
- The next session simulates the roll of die 10 times:

>>> import random

>>> for roll in range(10):

print(random.randint(1, 6), end = " ")

2 4 6 4 3 2 3 6 2 2

3. Formatting Text for output

- Many data-processing applications require output that has a **tabular format,** like that used in spreadsheets or tables of numeric data.
- In this format, numbers and other information are aligned in columns that can be either left-justified or right-justified.
  - A column of data is left-justified if its values are vertically aligned beginning with their leftmost characters.

- A column of data is right-justified if its values are vertically aligned beginning with their rightmost characters.

- The total number of data characters and additional spaces for a given datum in a formatted string is called its **field width.**

3.1 FORMATTING STRING:

- The simplest form of this operation is the following:

  <format string> % <datum>

- The following example shows how to right-justify and left- justify the string "four" within a field width of 6:

```
>>> "%6s" % "four"          # Right justify

' four'

>>> "%-6s" % "four“         # Left justify

'four '
```

- When the field width is positive, the datum is right-justified; when the field width is negative, you get left-justification.
- If the field width is less than or equal to the datum's print length in characters, no justification is added.
- The % operator works with this information to build and return a formatted string.

3.2 FORMATTING INTEGERS:

- To format integers, you use the letter d instead of s.
- To format a sequence of data values, you construct a format string that includes a format code for each datum and place the data values in a tuple following the % operator.
- The form of the second version of this operation follows:

  <format string> % (<datum–1>, ..., <datum–*n*>)

WITHOUT FORMATTING (integers):

```
>>> for exponent in range(7, 11):
        print(exponent, 10 ** exponent)
```

Surya Lakshmi Kantham Vinti
CSE DEPT
ACET

7 10000000

8 100000000

9 1000000000

10 10000000000

WITH FORMATTING (integers):

- The first column is left-justified in a field width of 3, and the second column is right-justified in a field width of 12.

```
>>> for exponent in range(7, 11):
        print("%-3d%12d" % (exponent, 10 ** exponent))
7          10000000
8         100000000
9        1000000000
10  10000000000
```

3.3 FORMATTING FLOAT:

- The format information for a data value of type float has the form

  %<field width>.<precision>f

  where .<precision> is optional.

- Example of the use of a format string, which says to use a field width of 6 and a precision of 3 to format the float value 3.14:

```
>>> "%6.3f" % 3.14
' 3.140'
```

- Note that Python adds a digit of precision to the string and pads it with a space to the left to achieve the field width of 6. This width includes the place occupied by the decimal point.

4. Strings

- E-mail, text messaging, Web pages, and word processing all rely on and manipulate data consisting of strings of characters.

- A string is a sequence of characters enclosed in single or double quotation marks.

- The following session with the Python shell shows some example strings:

>>> 'Hello there!'

'Hello there!'

>>> "Hello there!"

'Hello there!'

>>> ''

''

>>> """"

''

4.1 Accessing Character and Substring in Strings

The Structure of Strings:

- A string is a **data structure** i.e., it is a compound unit that consists of several other pieces of data.
- A string is a sequence of zero or more characters.
- The string is an **immutable data structure**. This means that its internal data elements, the characters, can be accessed, but cannot be replaced, inserted, or removed.
- A string's length is the number of characters it contains. Python's **len** function returns this value when it is passed a string, as shown in the following session:

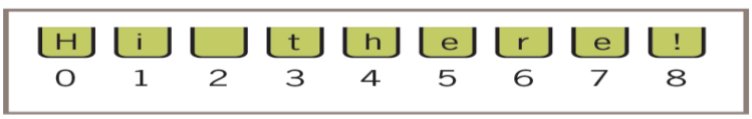
  >>> **len("Hi there!")**

  9

  >>> len("")

  0

- The positions of a string's characters are numbered from 0, on the left, to the length of the string minus 1, on the right.
- The following figure illustrates the sequence of characters and their positions in the string "Hi there!".
- Note that the ninth and last character, '!', is at position 8.



The Subscript Operator:

- The **subscript operator []** inspects one character at a given position without visiting all the characters in a given string / sequence.
- The simplest form of the subscript operation is the following:

  <a string>[<integer or index>]

- The first part of this operation is the string you want to inspect.
- The integer in brackets is the index that indicates the position of a particular character in that string.

```
>>> name = "Alan Turing"
>>> name[0]                 # Examine the first character
'A'
>>> name[3]                 # Examine the fourth character
'n'
>>> name[len(name)]      # Oops! An index error!
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1] # Examine the last character
'g'
>>> name[-l]                # Shorthand for the last character
'g'
>>> name[-2]                # Shorthand for next to last character
'n'
```

- The next code segment uses a count-controlled loop to display the characters and their positions:

```
>>> data = "Hi there!"
>>> for index in range(len(data)):
                print(index, data[index])
```

Surya Lakshmi Kantham Vinti
CSE DEPT
ACET

0 H

1 i

2

3 t

4 h

5 e

6 r

7 e

8 !

4.2 Slicing for Substrings :

- You can use Python's subscript operator to obtain a substring through a process called **slicing.**
- To extract a substring, the programmer places a colon (:) in the subscript. An integer value can appear on either side of the colon
- When two integer positions are included in the slice, the range of characters in the substring extends from the first position up to but not including the second position.
- When the integer is omitted on either side of the colon, all of the characters extending to the end or the beginning are included in the substring

```
>>> name="Surya Lakshmi"
>>> name[:]
'Surya Lakshmi'
>>> name[0:5]
'Surya'
>>> name[0:]
'Surya Lakshmi'
>>> name[4:7]
'a L'
name[-4:-1]
```

'shm'

4.3 Testing for a Substring with the in Operator:

- When used with strings, the left operand of in is a target substring, and the right operand is the string to be searched.
- The operator in returns True if the target string is somewhere in the search string, or False otherwise.

```
name="Surya Lakshmi"
>>> "z" in name
False
>>> "u" in name
True
>>> "Lak" in name
True
```

5. Strings and Number Systems

- The system used to represent numbers are called number systems. Various number systems are:
  - decimal number system(base ten number system)
  - binary number system (base two number system)
  - octal number system (base eight number system)
  - hexadecimal number system (base 16 number system)
- To identify the system being used, you attach the base as a subscript to the number.
- For example, the following numbers represent the quantity 41510 in the binary, octal, decimal, and hexadecimal systems:
  - 415 in binary notation 1100111112
  - 415 in octal notation 6378
  - 415 in decimal notation 41510
  - 415 in hexadecimal notation $19F16$

The Positional System for Representing Numbers

- All of the number systems we have examined use **positional notation**—that is, the value of each digit in a number is determined by the digit's position in the number.
- In other words, each digit has a **positional value**. The positional value of a digit is determined by raising the base of the system to the power specified by the position (***base position)***.
- To determine the quantity represented by a number in any system from base 2 through base 10, you multiply each digit (as a decimal number) by its positional value and add the results.
- The following example shows how this is done for a three-digit number in base 10:

$$
\begin{aligned}
&415_{10} = \\
&4 * 10^2 + 1 * 10^1 + 5 * 10^0 = \\
&4 * 100 + 1 * 10 + 5 * 1 = \\
&400 + 10 + 5 = 415
\end{aligned}
$$

Converting Binary to Decimal

- Each digit or bit in a binary number has a positional value that is a power of 2.
- We occasionally refer to a binary number as a string of bits or a **bit string.**
- Determine the integer quantity that a string of bits represents in the usual manner: Multiply the value of each bit (0 or 1) by its positional value and add the results

$$
\begin{aligned}
&1100111_2 = \\
&1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = \\
&1 * 64 + 1 * 32 + 0 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 1 * 1 = \\
&64 + 32 + 4 + 2 + 1 = 103
\end{aligned}
$$

- **Python script to convert binary number to decimal number**

```
bitString = input("Enter a string of bits: ")
decimal = 0
exponent = len(bitString) - 1
for digit in bitString:
```

Surya Lakshmi Kantham Vinti
CSE DEPT
ACET

```
        decimal = decimal + int(digit) * 2 ** exponent
        exponent = exponent - 1
    print("The integer value is", decimal)
```

OUTPUT :

Enter a string of bits: 1111

The integer value is 15

Enter a string of bits: 101

The integer value is 5

Converting Decimal to Binary:

- This algorithm repeatedly divides the given decimal number by 2.
- After each division, the remainder (either a 0 or a 1) is placed at the beginning of a string of bits.
- The quotient becomes the next dividend in the process. The string of bits is initially empty, and the process continues while the decimal number is greater than 0.

Python script to convert decimal number to binary number

```
decimal = int(input("Enter a decimal integer: "))
if (decimal == 0):
        print(0)
else:
        print("Quotient Remainder Binary")
        bitString = ""
        while decimal > 0:
                remainder = decimal % 2
                decimal = decimal // 2
                bitString = str(remainder) + bitString
                print("%5d%8d%12s" % (decimal, remainder,bitString))
        print("The binary representation is", bitString)
```

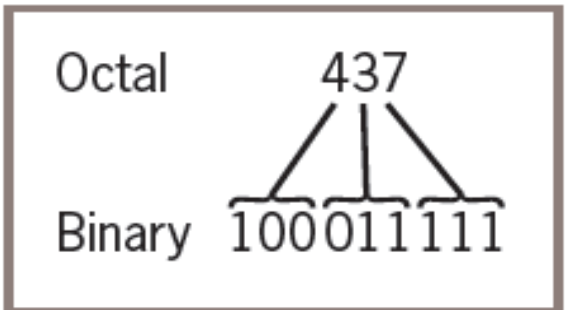
Enter a decimal integer: 34

Quotient	Remainder	Binary
17	0	0
8	1	10
4	0	010
2	0	0010
1	0	00010
0	1	100010

The binary representation is 100010

Octal Numbers

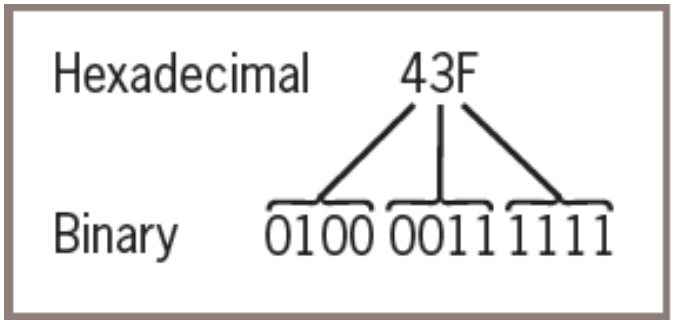
- The **octal system** uses a base of eight and the digits 0 . . . 7.
- Conversions of octal to decimal and decimal to octal use algorithms similar to those discussed thus far (using powers of 8 and multiplying or dividing by 8, instead of 2).
- To convert binary to octal, you begin at the right and factor the bits into groups of three bits each. You then convert each group of three bits to the octal digit they represent.



Hexadecimal Numbers:

- The **hexadecimal** or base-16 system (called "hex" for short), which uses 16 different digits, provides a more concise notation than octal for larger numbers.
- Base 16 uses the digits 0 . . . 9 for the corresponding integer quantities and the letters A . . . F for the integer quantities 10 . . . 15.
- The conversion between numbers in the two systems works as follows.
- Each digit in the hexadecimal number is equivalent to four digits in the binary number.
- Thus, to convert from hexadecimal to binary, you replace each hexadecimal digit with the corresponding 4-bit binary number.

- To convert from binary to hexadecimal, you factor the bits into groups of four and look up the corresponding hex digits.



5 String Methods :

String Method	What it Does
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s. startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

Examples :

```
>>> s = "Hi there!"

>>> len(s)
```

```
9
>>> s.center(11)
' Hi there! '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find("the")
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> " ".join(words)
'Hithere!'
>>> " ". join(words)
'Hi there!'
>>> s.lower()
'hi there!'
>>> s.upper()
```

'HI THERE!'

>>> s.replace('i', 'o')

'Ho there!'

>>> " Hi there! ".strip()

'Hi there!'

6) Data Encryption

- Data travelling on the Internet is vulnerable to spies and potential thieves.
- It is easy to observe data crossing a network, particularly now that more and more communications involve wireless transmissions.
  - For example, a person can sit in a car in the parking lot outside any major hotel and pick up transmissions between almost any two computers if that person runs the right **sniffing software**
  - **Data encryption** is a security method where information is encoded and can only be accessed or decrypted by a user with the correct encryption key.
- The sender encrypts a message by translating it to a secret code, called a **cipher text**.
- At the other end, the receiver decrypts the cipher text back to its original **plaintext** form.
- Both parties to this transaction must have at their disposal one or more keys that allow them to encrypt and decrypt messages.

6.1 Caesar cipher:

- This encryption strategy replaces each character in the plaintext with the character that occurs a given distance away in the sequence.
- For positive distances, the method wraps around to the beginning of the sequence to locate the replacement characters for those characters near its end.
- For example, if the distance value of a Caesar cipher equals three characters, the string "invaders" would be encrypted as “lqydghuv“
- To decrypt this cipher text back to plaintext, you apply a method that uses the same distance value but looks to the left of each character for its replacement.
- This decryption method wraps around to the end of the sequence to find a replacement character for one near its beginning

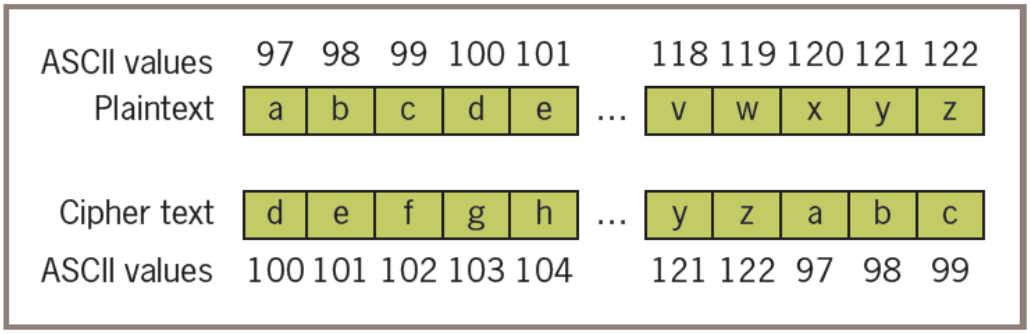


Figure 4-2 A Caesar cipher with distance +3 for the lowercase alphabet

Python Script to encrypt plain text to cipher text using Caesar Cipher

```
plainText = input("Enter a one-word, lowercase message: ")
distance = int(input("Enter the distance value: "))
code = ""
for ch in plainText:
    ordvalue = ord(ch)
    cipherValue = ordvalue + distance
    if cipherValue > ord('z'):
        cipherValue = ord('a') + distance - (ord('z') - ordvalue + 1)
    code += chr(cipherValue)
print(code)
```

OUTPUT :

```
>>>Enter a one-word, lowercase message: python
Enter the distance value: 3
sbwkrq
>>>Enter a one-word, lowercase message: xyz
Enter the distance value: 3
abc
```

Python Script to encrypt cipher text to plain text using Caesar Cipher

```
code = input("Enter the coded text: ")
distance = int(input("Enter the distance value: "))
plainText = ""
for ch in code:
    ordvalue = ord(ch)
    cipherValue = ordvalue - distance
    if cipherValue < ord('a'):
        cipherValue = ord('z') - (distance + (ord('a') - ordvalue - 1))
    plainText += chr(cipherValue)
print(plainText)
```

OUTPUT :

```
>>>Enter the coded text: sbwkrq
Enter the distance value: 3
python
>>>Enter the coded text: abc
Enter the distance value: 3
xyz
```

- The main shortcoming of this encryption strategy is that the plaintext is encrypted one character at a time, and each encrypted character depends on that single character and a fixed distance value.
- In a sense, the structure of the original text is preserved in the cipher text, so it might not be hard to discover a key by visual inspection.

6.2 Block cipher:

- A block cipher uses plaintext characters to compute two or more encrypted characters.
- This is accomplished by using a mathematical structure known as an **invertible matrix to determine the values of** the encrypted characters.

- The matrix provides the key in this method. The receiver uses the same matrix to decrypt the cipher text.
- The fact that information used to determine each character comes from a block of data makes it more difficult to determine the key