# Day-2: Control Statement and String Manipulation 22/06/2018

# Workshop on "Introduction to Python"

Department Of Computer Science Engineering

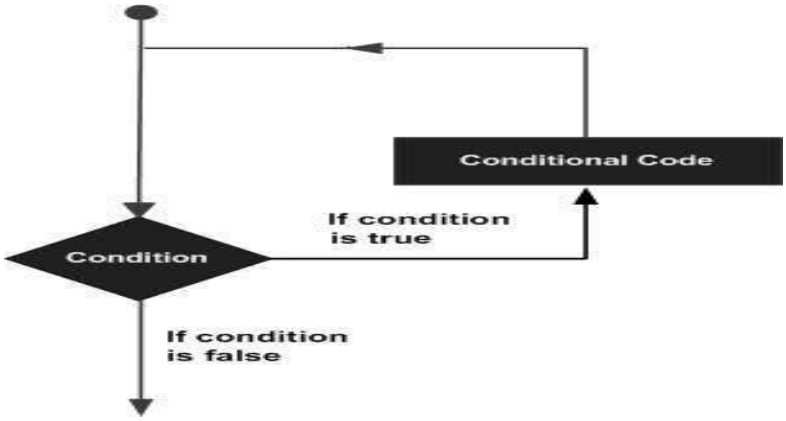
SKFGI, Mankundu

# Control 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.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –




Python programming language provides following types of loops to handle looping requirements.

| Sl No. | Loop Type & Description |
|---|---|
| **1** | **while loop**<br>Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| **2** | **for loop**<br>Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| **3** | **nested loops**<br>You can use one or more loop inside any another while, for or do..while loop. |

# 1.While loop:-

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true. The **syntax** of a **while** loop in Python programming language is –

**while expression:**

    **statement(s)**

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

**Program-1:**

```
count = 0
while count < 9:
   print ('The count is:', count)
   count = count + 1
print ("Exit from loop")
```

**output:-**

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Exit from loop
```

**Program-2**

```
print("enter the number")
x=(int)(input())
f=1
while x > 0:
    f=f * x
    x= x - 1
print(f)
```

**output:-**

```
enter the number
6
factorial is  720
```

**Program-3**

```
print("enter the number")
x=(int)(input())
rev=0
while x > 0:
    r=x % 10
    rev=rev*10+r
    x= x // 10
print("reverse is",rev)
```

**output:-**

```
enter the number
123
reverse is 321
```

**Program-4**

```
print("enter the number")
x=(int)(input())
f=x
rev=0
while x > 0:
```

```
    r=x % 10
    rev=rev*10+r
    x= x // 10
print("reverse is",rev)
if (f == rev):
   print(f ," is palindrome")
else:
   print(f ," is not palindrome")
```

**output:-**

```
enter the number
121
reverse is 121
121  is palindrome
```

**output:-**

```
enter the number
123
reverse is 321
123  is not palindrome
```

**Program-5**

```
#Fibonacii Series upto n terms
print("enter the value of n")
n=(int)(input())
a=0
b=1
if(n==1):
    print(a)
if(n==2):
    print(a)
    print(b)
if(n>2):
    i=1
```

```
    print(a)
    print(b)
    while i<=n-2:
        c=a+b
        a=b
        b=c
        print(c)
        i=i+1
```

**output:-**

```
enter the value of n
10
0
1
1
2
3
5
8
13
21
34
```

## 2.For loop:-

**For** loop is used for iterating over a sequence (that is either a list, a tuple or a string). To loop through a set of code a specified number of times, we can use the **range()** function which returns a sequence of numbers, starting **from 0 by default, and increments by 1** (by default), and ends at a specified number.

**Program-1**

```
for x in range(6):
      print(x)
```

**output:-**

0

1

2

3

4

5

**Program-2**

```
for x in range(2,6):
        print(x)
```

**output:-**

2

3

4

5

**Program-3**

```
for x in range(2,16,3):
         print(x)
```

**output:-**

2

5

8

11

14

**Program-4**

```
print("enter the range")
x=(int)(input())
for i in range(1,x,2):
    print(i)
```

**output:-**

enter the range

10

1

3

5

7

9

**Program-5**

```
print("enter the range")
x=(int)(input())
f=1
for i in range(1,x+1):
    f=f*i
print(f)
```

**output:-**

enter the range

5

120

**Program-6**

```
print("enter the number")
x=(int)(input())
rev=0
for i in range(x,0,-1):
    r=x%10
    rev=rev*10+r
    x=x//10
    if x==0:
        break
print(rev)
/*alternative*/
reverse = ''
num = input()
```

```
for i in range(len(num), 0, -1):
    reverse += num[i-1]
print(int(reverse))
```

**output:-**

```
enter the number
123
321
```

**Nested loop:-**

**Program-1**

```
*
* *
* * *
* * * *
print("enter the number")
n=(int)(input())
for i in range(1,n+1):
    for j in range(1,i+1):
        print("*",end=" ")
    print("")
```

**Program-2**

```
print("enter the number")
n=(int)(input())
for i in range(n,1,-1):
    for j in range(1,i,1):
        print("*",end=" ")
    print("")
```

**output:-**

**enter the number**

**5**

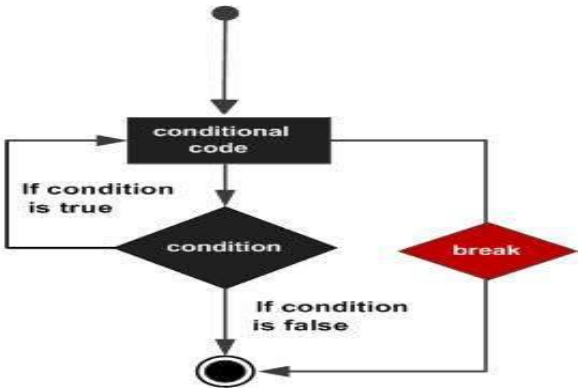
*** * * ***

* * *

* *

*

**3.Break statement:-**It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C. The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops. If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.




**Program-1**

```
for x in "Python":
   if( x == 'h'):
      break
   print("Current Letter :", x)
```

**output:-**

Current Letter : P

Current Letter : y

Current Letter : t

**Program-2**

```
a = 10
while a > 0:
   print ('value of a is:', a)
```

```
    a = a -1
    if (a == 5):
        break
print ("Happy ending!")
```

**output:-**

```
value of a is: 10
value of a is: 9
value of a is: 8
value of a is: 7
value of a is: 6
Happy ending!
```

**4. Continue statement**:- It returns the control to the beginning of the while loop.. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

**Program-1**

```
for i in range(0,10):
    if i==5:
        continue
    print(i)
print("end")
```

**output:-**

```
0
1
2
3
4
6
7
```

8

9

end

**Program-2**

```
for a in 'Python':
   if (a == 'h'):
      continue
   print ('Current Letter :', a)
```

**output:-**

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : o

Current Letter : n

**5.Pass statement:-** The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. However, nothing happens when pass is executed. Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So we use the pass statement to construct a body that does nothing.

**Program-1**

```
for a in 'Python':
   if (a == 'h'):
      pass
      print ('This is pass block')
   print ('Current Letter :', a)

print ("Happy ending!")
```

**output:-**

Current Letter : P

Current Letter : y

Current Letter : t

This is pass block

Current Letter : h

Current Letter : o

Current Letter : n

Happy ending!

# String:-

Python string is a built-in type text sequence. It is used to handle textual data in python. Python Strings are immutable sequences of Unicode points. Creating Strings are simplest and easy to use in Python.

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

**Note:** String are **immutable** which means that once created they can not changed. Whenever we try to modify an existing string variable , a new string created .

```
s1 ="listen"
print(s1,id(s1))
s1+= "silent"
print(s1,id(s1))
```

**Output:**

listen 35714656

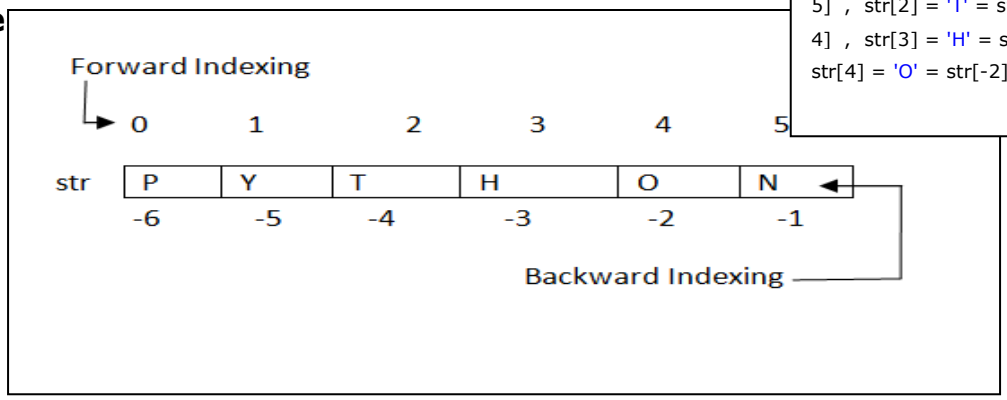
listensilent 36958064

**6.Accessing Python Strings**

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions (forward and backward).
- Both forward as well as backward indexing are provided using Strings in Python.
    - Forward indexing starts with 0,1,2,3,....

- Backward indexing starts with -1,-2,-3,-4,....

**Example**




str[0]='P'=str[-6] , str[1]='Y' = str[-5] , str[2] = 'T' = str[-4] , str[3] = 'H' = str[-3]
str[4] = 'O' = str[-2] , str[5] = 'N' = str[-1].

**Program-1**

```
var1 = 'Hello World!'
var2 = "Python Programming"
print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
```

**output:-**

```
var1[0]:  H
var2[1:5]:  ytho
```

**Program-2** program to retrieve String in reverse as well as normal form.

```
name="Rajat"
length=len(name)
i=0
for n in range(-1,(-length-1),-1):
    print name[i],"\t",name[n]
    i+=1
```

**Output:**

```
R       t
a       a
j       j
a       a
t       R
>>>
```

**7. Python Strings Operators**

To perform operation on string, Python provides basically 3 types of Operators that are given below.

1. Basic Operators.
2. Membership Operators.

3. Relational Operators.

### 7.1Python String Basic Operators

There are two types of basic operators in String "+" and "*".

#### *7.1.1String Concatenation Operator (+)*

The concatenation operator (+) concatenates two Strings and creates a new String.

*Python String Concatenation Example*

```
>>> "ratan" + "jaiswal"
```

**Output:**

```
'ratanjaiswal'
>>>
```

| Expression | Output |
|---|---|
| '10' + '20' | '1020' |
| "s" + "007" | 's007' |
| 'abcd123' + 'xyz4' | 'abcd123xyz4' |

**NOTE: Both the operands passed for concatenation must be of same type, else it will show an error.**

**Eg:**

```
'abc' + 3
>>>
```

**output:**

```
Traceback (most recent call last):
  File "", line 1, in
    'abc' + 3
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

#### 7.1.2Python String Replication Operator (*)

Replication operator uses two parameters for operation, One is the integer value and the other one is the String argument.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

**Python String Replication Example**

```
>>> print(5*"Vimal"  )
```

**Output:**

```
'VimalVimalVimalVimalVimal'
```

| Expression | Output |
|---|---|
| "soono"*2 | 'soonosoono' |
| 3*'1' | '111' |
| '$'*5 | '$$$$$' |

**NOTE: We can use Replication operator in any way i.e., int * string or string * int. Both the parameters passed cannot be of same type.**

## 7.2Python String Membership Operators

Membership Operators are already discussed in the Operators section. Let see with context of String.

**There are two types of Membership operators**

**1) in:**"in" operator returns true if a character or the entire substring is present in the specified string, otherwise false.

**2) not in:**"not in" operator returns true if a character or entire substring does not exist in the specified string, otherwise false.

*Python String membership operator Example*

```
>>> str1="javatpoint"
>>> str2='sssit'
>>> str3="seomount"
```

```
>>> str4='java'
>>> st5="it"
>>> str6="seo"
>>> str4 in str1
True
>>> str5 in str2
>>> st5 in str2
True
>>> str6 in str3
True
>>> str4 not in str1
False
>>> str1 not in str4
True
```

### 7.3 Python Relational Operators

All the comparison (relational) operators i.e., (<,><=,>=,==,!=) are also applicable for strings. The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

**Python Relational Operators Example**

```
>>> "RAJAT"=="RAJAT"
True
>>> "afsha">='Afsha'
True
>>> "Z"<>"z"
True
'sayon'>'sazon'
False
```

**Explanation:**

The ASCII value of a is 97, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 67 and so on. The comparison between strings are done on the basis on ASCII value.

Python String Slice Notation

Python String slice can be defined as a substring which is the part of the string. Therefore further substring can be obtained from a string.

There can be many forms to slice a string, as string can be accessed or indexed from both the direction and hence string can also be sliced from both the directions.

### 8.1.Python String Slice Syntax

```
<string_name>[startIndex:endIndex],
<string_name>[:endIndex],
<string_name>[startIndex:]
```

*Python String Slice Example 1*

```
>>> str="Nikhil"
>>> str[0:6]
'Nikhil'
>>> str[0:3]
'Nik'
>>> str[2:5]
'khi'
>>> str[:6]
'Nikhil'
>>> str[3:]
'hil'
```

**Note: startIndex in String slice is inclusive whereas endIndex is exclusive.**

String slice can also be used with Concatenation operator to get whole string.

### 8.2Specififying stride while slicing string:

In the slice operation, we can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string. The default value is stride is 1.

**Program-1**

```
str="welcome to the world of python"

print(str[2:15])

print(str[2:15:1])

print(str[2:105:2])

print(str[2:15:3])
```

**output:-**

```
lcome to the
lcome to the
loet h ol fpto
lmtt
```

**Program-2** reverse of a string using slice operation

```
str="welcome to the world of python"
print(str[::-1])
```

**output:-**

```
nohtyp fo dlrow eht ot emoclew
```

**Program-3** wap to check a string is palindrome or not

```
str="madam"
s2=str[::-1]
if str==s2:
    print("palindrome")
else:
    print("not")
```

**output:-**

```
palindrome
```

## 9.Built-in String Methods

**9.1.isnumeric():-**Returns true if a unicode string contains only numeric characters and false otherwise.

**Program-1**

```
str = u"this2009"
print (str.isnumeric())
str = u"23443434"
print (str.isnumeric())
```

**output:-**

```
False
True
```

**9.2.capitalize():-**

It returns a copy of the string with only its first character capitalized.

**Program-1**

```
str = "this is string example....wow!!!"
print ("output is: ", str.capitalize())
```

**output:-**

output is:  This is string example....wow!!!

**9.3. count(string,begin,end):-**

The method **count()** returns the number of occurrences of substring sub in the range [start, end].

**Program-1**

```
str = "this is string example....wow!!!"
sub = "i"
print ("output is : ", str.count(sub, 4, 40))
```

**output:-**

output is :  2

**Program-2**

```
str = "this is string example....wow!!!"
sub = "wow"
print ("output is : ", str.count(sub))
```

**output:-**

output is :  1

**9.4. find(substring ,beginIndex, endIndex):**

It determines if the specified string to be searched is present or not and it returns the index if it is found otherwise -1.

**Program-1**

```
str1 = "this is string example"
str2 = "exam"
print (str1.find(str2))
print (str1.find(str2, 10))
print (str1.find(str2, 40))
```

**output:-**

15

15

-1

**9.5. isalnum():-**

It checks whether the string consists of alphanumeric characters. This method returns true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

**Program-1**

```
str = "abcd2009"
print (str.isalnum())
str = "abcd ef"
print (str.isalnum())
```

**output:-**

```
True
False
```

**9.6**. **isalpha():-**This method returns true if all characters in the string are alphabetic and there is at least one character, false otherwise.

**Program-1**

```
str = "this"
print (str.isalpha())
str = "this is"
print (str.isalpha())
```

**output:-**

```
True
False
```

**9.7. isdigit():-**This method returns true if all characters in the string are digits and there is at least one character, false otherwise.

**Program-1**

```
str = "123456"
print (str.isdigit())
str = "1abc"
print (str.isdigit())
```

**output:-**

```
True
False
```

**9.8. islower():-**This method returns true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

**Program-1**

```
str = "THIS "
print (str.islower())
str = "this "
print (str.islower())
```

**output:-**

```
False
True
```

**9.9. upper():-**This method returns a copy of the string in which all case-based characters have been uppercased.

**Program-1**

```
str = "abcd"
print ("output is: ", str.upper())
```

**output:-**

```
output is:  ABCD
```

**9.10. swapcase()** :- This method returns a copy of the string in which all the case-based characters have had their case swapped.

**Program-1**

```
str = "this is STRING"
print (str.swapcase())
```

**output:-**

```
THIS IS string
```

**9.11. min():-**This method returns the  min alphabetical character from the given string.

**Program-1**

```
str = "bass"
print ("Min character: " + min(str))
```

**output:-**

Min character: a

**9.12. max():-**This method returns the max alphabetical character from the given string .

**Program-1**

```
str = "bass"
print ("Max character: " + max(str))
```

**output:-**

```
Max character: s
```

**9.13. replace()**:-It replaces a string with another string.

**Program-1**

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

**output:-**

```
Jello, World!
```

**9.14 endswith(suffix ,begin=0,end=n):**

It returns a Boolean value if the string terminates with given suffix between begin and end.

**Program-1**

```
string1="Welcome to SSSIT"
substring1="IT"
substring2="to"
print (string1.endswith(substring1)  )
print (string1.endswith(substring2,2,16))
```

**Output:**

```
>>>
True
False
```

**9.15 isupper():-**It returns False if characters of a string are in Upper case, otherwise False.

**Program-1**

```
    string1="Hello Python"
```

```
print (string1.isupper())
string2="WELCOME TO"
print (string2.isupper())
```

**Output:**

```
>>>
False
True
```

**9.16. isspace():** It returns True if the characters of a string are whitespace, otherwise false.

**Program-1**

```
string1="    "
print (string1.isspace())
string2="WELCOME TO WORLD OF PYT"
print (string2.isspace())
```

**Output:**

```
>>>
True
False
>>>
```

**9.17 len(string):-**

It returns the length of a string.

**Program-1**

```
string1="    "
print (len(string1))
string2="WELCOME TO SSSIT"
print (len(string2))
```

**Output:**

```
>>>
4
16
>>>
```

**9.18 lower()**:-

It converts all the characters of a string to Lower case.

**Program-1**

```
string1="Hello Python"
print (string1.lower() )
string2="WELCOME TO SSSIT"
print (string2.lower())
```

Output:

```
>>>
hello python
welcome to sssit
```

```
>>>
```

**9.19: startswith(string,start index, end index):-**

This method returns a Boolean value if the string starts with given str between begin and end.

**Program-1**

```
string1="Hello Python"
print (string1.startswith('Hello'))
string2="welcome to SSSIT"
print (string2.startswith('come',3,7))
```

**Output:**

```
>>>
True
True
>>>
```

**9.20 lstrip():-**

It removes all leading whitespace of a string and can also be used to remove particular character from leading.

**Program-1**

```
string1="    Hello Python"
print (string1.lstrip())
string2="@@@@@@@@welcome to SSSIT"
print (string2.lstrip('@'))
```

**Output:**

```
>>>
Hello Python
welcome to world to SSSIT
>>>
```

**9.21rstrip() :-**

It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

**Program-1**

```
string1="    Hello Python     "
print (string1.rstrip() )
string2="@welcome to SSSIT!!!"
print (string2.rstrip('!') )
```

**Output:**

```
>>>
                Hello Python
@welcome to SSSIT
>>>
```

**9.22strip():-**It removes all trailing and leading whitespace of a string and can also be used to remove particular character from trailing.

**Program-1**

```
string2="!!!welcome to SSSIT!!!"
print (string2.strip('!')    )
```

**Output:** welcome to SSSIT

**9.23 title():-** returns string in title case(capitalized each word )

**Program-1**

```
str=”the world is beautiful”
print(str.title())
```

**output:** The World Is Beautiful

**9.24 split():-** returns a list of substring separated by the specified delimiter. If no delimiters is specified then by default it split string on all whitespace character

**Program-1**

```
str="abc,def,ghi,jkl"
print(str.split(','))
```

**Sample Program:-**

**output:** ['abc', 'def', 'ghi', 'jkl']

**9.25 join(list):-** it is just the opposite of spilt. The function joins a list of string using the delimiter with which the function is invoked

**Program-1**

```
l=['abc', 'def', 'ghi', 'jkl']
print(','.join(l))
```

**output:** abc,def,ghi,jkl

**9.26 enumerate(str):-** returns an enumerate object that list the index and value of all characters in the string as pair

**Program-1**

```
str="hello"
print(list(enumerate(str)))
```

**output:** [(0, 'h'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o')]

**9.27 ljust(width, fillchar):-** returns a string left-justified to a local of width column. Column without characters are padded with the character specified in the fill char argument.

**Program-1**

```
str="hello"
print(str.ljust(10,"*"))
```

**output:** hello*****

**9.28 rjust(width, fillchar):-** returns a string right-justified to a local of width column. Column without characters are padded with the character specified in the fill char argument.

**Program-1**

```
str="hello"
print(str.rjust(10,"*"))
```

**output:** *****hello

**9.29 Center (width, fillchar):-** returns a string with the original string centered to a total width columns and filled with fillchar in column that not have character

**Program-1**

```
str="hello"
print(str.center(10,"*"))
```

**output:** **hello***

**1. Program to check whether a string is Palindrome or not.**

```
my_str = 'MadAm'
my_str = my_str.casefold()
```

```
rev_str = reversed(my_str)
if list(my_str) == list(rev_str):
   print("It is palindrome")
else:
   print("It is not palindrome")
```

**output:-**

```
It is palindrome
```

2. **Program to sort alphabetically the words form a string provided by the user**

```
my_str = input("Enter a string: ")
words = my_str.split()
words.sort()
print("The sorted words are:")
for word in words:
   print(word)
```

**output:-**

```
Enter a string: my college name is skfgi
The sorted words are:
college
is
my
name
skfgi
```

3. **Identify the larger string between two strings taken as user input:-**

```
string1=input("Enter first string:")
string2=input("Enter second string:")
count1=0
count2=0
for i in string1:
```

```
    count1=count1+1
for j in string2:
    count2=count2+1
if(count1<count2):
    print("Larger string is:")
    print(string2)
elif(count1==count2):
    print("Both strings are equal.")
else:
    print("Larger string is:")
    print(string1)
```

**output:-**

```
Enter first string:abc
Enter second string:ABCD
Larger string is:
ABCD
```

4. **Check if a substring is present in a given string:-**

```
string = "college name skfgi"
sub_str ="college"
if (string.find(sub_str) == -1):
        print("NO")
else:
        print("YES")
```

**output:-**

```
YES
```

5. **Check if two strings are anagram or not:-**

An anagram of a string is another string that contains same characters, only the order of characters can be different. sorted() is an

inbuilt function which does not modify the original string, but returns sorted string.

```
s1 ="listen"
s2 ="silent"
if(sorted(s1)== sorted(s2)):
        print("The strings are anagrams.")
else:
        print("The strings aren't anagrams.")
```

**output:-**

```
The strings are anagrams.
```

6. **To count the number of vowels and digits in a string:-**

```
string=input("Enter string:")
vowels=0
count1=0
for i in string:
      if(i=='a' or i=='e' or i=='i' or i=='o' or i=='u' or i=='A' or i=='E'
or i=='I' or i=='O' or i=='U'):
            vowels=vowels+1
      if(i.isdigit()):
           count1=count1+1
print("Number of vowels are:")
print(vowels)
print("The number of digits is:")
print(count1)
```

**output:-**

```
Enter string:abcd1234
Number of vowels are:
1
The number of digits is:
4
```

7. **WAP that encrypts a message by adding a key value to every character (Caesar cipher)[take key=3]**

```
str="Hello"
index=0
s=""
while index<len(str):
    l=str[index]
    s+=chr(ord(l)+3)
    index+=1
print(s)
```

**output:-**

Khoor

8. **WAP to check A PAN number format is valid or not**

```
s=input('enter the PAN Card number')
s1=s[0:5]
s2=s[5:9]
s3=s[9]
if s1.isalpha()==True and s2.isdigit()== True and s3.isalpha()==True:
    print("correct format of pan number")
else:
    print("not a valid pan number")
```

**output:-**

enter the PAN Card number 120aumpg5d

not a valid pan number

# 10.String Format (.format)

Python's str.format() method of the string class allows you to do variable substitutions and value formatting.

Formatters work by putting in one or more **replacement fields** or placeholders — defined by a pair of curly braces {} — into a string and calling the str.format() method. You'll pass into the method the value you

want to concatenate with the string. This value will be passed through in the same place that your placeholder is positioned when you run the program. Let's print out a string that uses a formatter:

1. print("Sammy has {} balloons.".format(5))
   *output:* Sammy has 5 balloons.
2. open_string = "Sammy loves {}."
   print(open_string.format("open source"))
   *output:* Sammy loves open source.
   When we leave curly braces empty without any parameters, Python will replace the values passed through the str.format() method in order. As we have seen, so far, a formatter construction with two empty curly braces with two values passed through will look like this:
3. sammy_string = "Sammy loves {} {}, and has {} {}."
   print(sammy_string.format("open-source", "software", 5, "balloons"))
   #Pass 4 strings into method
   *output:* Sammy loves open-source software, and has 5 balloons.
4. We can pass these index numbers into the curly braces that serve as the placeholders in the original string:
   print("Sammy the {0} has a pet {1}!".format("shark", "pilot fish"))
   *output:* Sammy the shark has a pet pilot fish!
   print("Sammy the {1} has a pet {0}!".format("shark", "pilot fish"))
   *output:* Sammy the pilot fish has a pet shark!
5. In addition to positional arguments, we can also introduce keyword arguments that are called by their keyword name:
   print("Sammy the {0} {1} a {pr}.".format("shark", "made", pr = "pull request"))
   *output:* Sammy the shark made a pull request.

# Specifying Type

We can include more parameters within the curly braces of our syntax. We'll

use the format code syntax {field_name:conversion},
where field_name specifies the index number of the argument to
the str.format() method that we went through in the reordering section,
and conversion refers to the conversion code of the data type that you're using with the formatter.

The conversion type refers to the the single-character type code that Python uses. The codes that we'll be using here are s for string, d to display decimal integers (10-base), and f which we'll use to display floats with decimal places. You can read more about the Format-Specification Mini-Language through Python 3's official documentation.

6. print("Sammy ate {0:f} percent of a {1}!".format(75, "pizza"))
   *output:* Sammy ate 75.000000 percent of a pizza!
7. print("Sammy ate {0:s} percent of a {1}!".format('75', "pizza"))
   *output:* Sammy ate 75 percent of a pizza!
8. print("Sammy ate {0:d} percent of a {1}!".format(75, "pizza"))
   *output:* Sammy ate 75 percent of a pizza!
9. print("Sammy ate {0:b} percent of a {1}!".format(75, "pizza"))
   *output:* Sammy ate 1001011 percent of a pizza!
10. print("Sammy ate {0:e} percent of a {1}!".format(75, "pizza"))
    *output:* Sammy ate 7.500000e+01 percent of a pizza!
11. print("Sammy ate {0:.3f} percent of a pizza!".format(75.765367))
    *output:* Sammy ate 75.765 percent of a pizza!
12. print("Sammy ate {0:.0f} percent of a pizza!".format(75.765367))
    *output:* Sammy ate 76 percent of a pizza!

# Padding Variable Substitutions

We can add a number to indicate field size (in terms of characters) after the colon : in the curly braces of our syntax:

**13.** print("Sammy has {0:4} red {1:16}!".format(5, "balloons"))

*output:* Sammy has  5 red balloons  !

As we see, by default strings are left-justified within the field, and numbers are right-justified. You can modify this by placing an alignment code just following the colon. < will left-align the text in a field, ^ will center the text in the field, and > will right-align it.

**14.** print("Sammy has {0:<4} red {1:^16}!".format(5, "balloons")

*output:* Sammy has 5  red  balloons  !

By default, when we make a field larger with formatters, Python will fill the field with whitespace characters. We can modify that to be a different character by specifying the character we want it to be directly following the colon:

**15.** print("{:*^20s}".format("Sammy"))

*output:* *******Sammy********

We can combine these parameters with other parameters we've used before:

**16.** print("Sammy ate {0:10.3f} percent of a pizza!".format(75.765367))

*output:* Sammy ate  75.765 percent of a pizza!

# Using Variables

So far, we have passed integers, floats, and strings into the str.format() method, but we can also pass variables through the method. This works just like any other variable.

**17.** nBalloons = 8

print("Sammy has {} balloons today!".format(nBalloons))

*output:* Sammy has 8 balloons today!

# String format operator

One of Python's attractive features is the string format operator %.

Example:

**18.** print ("Great batsman %s has scored more than %d runs" % ("Sachin", 18000))

*output:*Great batsman Sachin has scored more than 18000 runs

19.

a=5

b=6

print("So you are %d of %d" % (a, b))

*output:* So you are 5 of 6

| **Symbol** | **Purpose** | **Symbol** | **Purpose** |
|---|---|---|---|
| %c | Char | %o | Octal |
| %d or %i | Signed Decimal Integer | %x or %X | Hexadecimal integer |
| %s | string | %e | Exponential notation |
| %f | Floating point number | %g | Short number in floating point/ exponential number |