
UNIT – III CONTROL FLOW, FUNCTIONS, STRINGS

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); **Iteration:** state, while, for, break, continue, pass;

Fruitful functions: return values, parameters, local and global scope, function composition, recursion; **Strings:** string slices, immutability, string functions and methods, string module; Lists as arrays.

Illustrative programs: square root, GCD, exponentiation, sum an array of numbers, linear search, binary search.

CONDITIONALS:

Boolean Values and Operators

Discuss Boolean values and operators in detail.

Boolean Expression

- It is an expression having two values either True or False to represent the truth values of logic and Boolean algebra.
- It uses the operator ==, which compares two operands and produces True if they are equal and False if they are not equal.

Example:

```
>>> 5 == 5
```

```
True
```

```
>>> 5 == 6
```

```
False
```

- True and False are special values that belong to the type bool; they are not strings:

```
>>> type(True)
```

```
<class 'bool'>
```

```
>>> type(False)
```

```
<class 'bool'>
```

Operators

Relational Operators

- It is used to compare the relation between two operands.
- The == operator is one of the **relational operators**.

- The Python symbols are different from the mathematical symbols.
- A common error is using a single equal sign (=) instead of a double equal sign (==).
- Here, = is an assignment operator and == is a relational operator.

Example

x == y	# x is equal to y
x != y	# x is not equal to y
x > y	# x is greater than y
x < y	# x is less than y
x >= y	# x is greater than or equal to y
x <= y	# x is less than or equal to y

Logical Operators

- There are three **logical operators**: and, or and not.
- For example, x > 0 and x < 10 is true, only if x is greater than 0 and less than 10.
- Finally, the not operator negates a Boolean expression, so not (x > y) is true if x > y is false.

2 Marks

1. What is Boolean Expression?

- It is an expression having two values either True or False to represent the truth values of logic and Boolean algebra.
- It uses the operator ==, which compares two operands and produces True if they are equal and False if they are not equal.

2. What is Relational Operator?

- It is used to compare the relation between two operands.
- The == operator is one of the **relational operators**.
- A common error is using a single equal sign (=) instead of a double equal sign (==).
- Here, = is an assignment operator and == is a relational operator.

3. What is Logical Operator?

- There are three **logical operators**: and, or and not.
- For example, x > 0 and x < 10 is true, only if x is greater than 0 and less than 10.
- Finally, the not operator negates a Boolean expression, so not (x > y) is true if x > y is false.

DECISION MAKING / CONDITIONAL STATEMENTS

Explain in detail about various conditional statements with example. (Or) Appraise with an example nested if and elif header in python. (Jan 2018) [Jan 2019] (or) Outline the conditional branching statements in Python with an example. [Jan 2022]

- A statement that controls the flow of execution depending on some condition.
- Python provides the following conditional statements or selection structures (decision making).
 1. if statement (Conditional)
 2. if...else statement (Alternative)
 3. if...elif...else statement (Chained Conditional)
 4. Nested Conditional

1. if statement (Conditional)

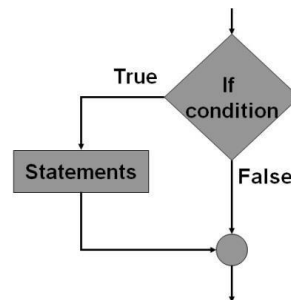
- It consists of Boolean expression followed by set of statements.
- It evaluates the expression and executes the statements only if the condition is true.

Syntax:

```
if Test Expression:  
    True Statements
```

Example:

```
a=10  
b=5  
if a>b:  
    print "a is greater than b"
```



Output:

a is greater than b

Compound Statement

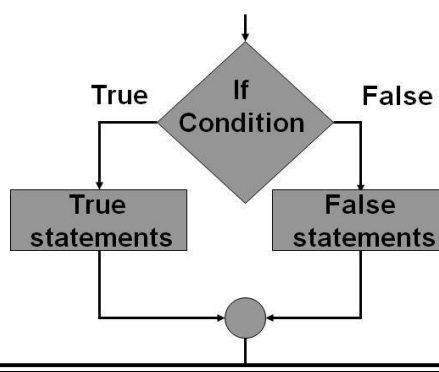
- A statement that consists of a header and a body.
- The header ends with a colon (:). The body is indented relative to the header.

2. if...else Statement (Alternative)

- A second form of if statement is "alternative execution".
- It is a two way decision making statement.
- It evaluates the expression if the condition is true it executes the true statements else it executes the false statements.

Syntax:

```
if Test Expression:  
    True Statements  
else:  
    False Statements
```



Example 1:

```

a=10
b=5
if a>b:
    print "a is greater than b"
else:
    print "a is less than b"

```

Output:

```

a is greater than b

```

Example 2:

```

x=3
if x % 2 == 0:
    print "x is even"
else:
    print "x is odd"

```

Output:

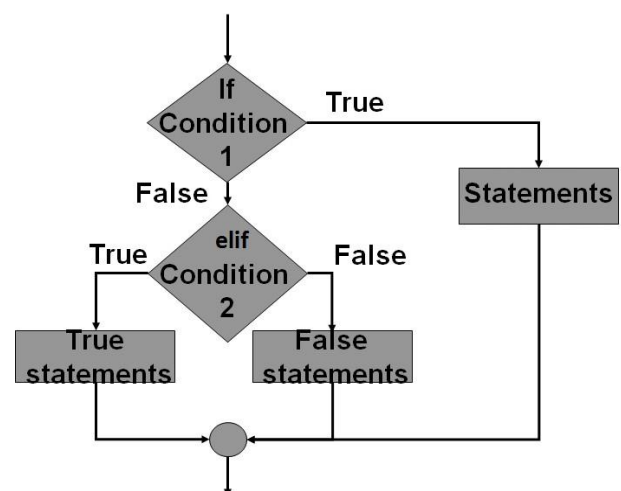
```

x is odd

```

3. if...elif...else Statement (Chained Conditional)

- It is a multi-way decision making statement.
- It is a conditional statement with a series of alternative branches.
- Each condition is checked in order. If the first condition is false, the next is checked and so on.
- If the condition is true, it executes the block and the statement ends.
- **elif** is an abbreviation of "**else if**". There is no limit on the number of elif statements.

**Syntax:**

```

if Test Expression 1:
    Statements
elif Test Expression 2:
    True Statements
else:
    False Statements

```

Example:

```

x=3
y=3

```

if $x < y$:

```
        print "x is less than y"
elif x > y:
    print "x is greater than y"
else:
    print "x and y are equal"
```

Output:

x and y are equal

4. Nested Conditionals

- One conditional (if...else) statement is nested within another conditional statement.

Example:

```
x=3
y=4
if x == y:
    print "x and y are equal"
else:
    if x < y:
        print "x is less than y"
    else:
        print "x is greater than y"
```

Output:

x is less than y

ITERATION (LOOPING)

Explain various looping statements in detail. (Or) Explain with an example while loop statement in python. (Jan 2018). (Or) Mention the different types of iterative structure allowed in python.[May 2019] (or) Outline while statement in Python with an example. [Jan 2022]

- The loop is defined as the block of statements which are executed repeatedly for certain number of times or until some condition is satisfied.
- There are two types of looping structures. They are,
 - for loop
 - while loop

State (Multiple Assignments)

- It is legal to make more than one assignment to the same variable.
- A new assignment makes an existing variable refer to a new value (and stop referring to the old value).

Example:

```
>>>a = 5
>>>print a
>>>a = 7
```

```
>>>print a
```

- In Python, an assignment statement can make two variables equal, but they don't have to stay that way:

Example:

```
>>>a = 5
>>>b = a    # a and b are now equal
>>>a = 3    # a and b are no longer equal
```

Updating Variables

- One of the most common forms of multiple assignments is an update, where the new value of the variable depends on the old.

```
>>>x = x+1
```

- This means “get the current value of x, add one, and then update x with the new value.”

Example:

```
>>> x = x+1
NameError: name 'x' is not defined
```

- Before you can update a variable, you have to initialize it, usually with a simple assignment:

```
>>> x = 0
>>> x = x+1
```

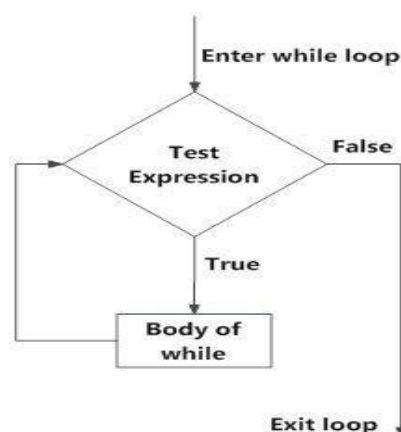
- Updating a variable by adding 1 is called an increment; subtracting 1 is called a decrement.

The while Loop [Jan 2022]

- The while loop is an entry controlled loop or top tested loop statement
- It means that, the condition is evaluated first, if it is true and then the body of the loop is executed.
- It is a repetitive control structure, used to execute the statements within the body until the condition becomes false.

Syntax

```
while Test Expression:
    Statement 1
    Statement 2
    .....
    Statement n
```



Example:

```
count=0
while count<=3:
    print "Python Programming"
    count=count+1
else:
    print "Exit"
```

Output:

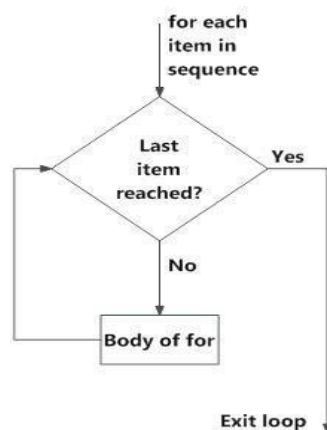
```
Python Programming
Python Programming
Python Programming
Python Programming
Exit
```

The for Loop

- It is used to repeat or execute a series of statements for specific number of times.
- The first line of the for statement is used to state how many times the code should be repeated.
- A **stepper variable** is used to count through each iteration of the loop.

Syntax

```
for variable in sequence:
    Statement 1
    Statement 2:
    .....
    Statement n
```



- Here, the variable stores all value of each item. Sequence may be a string, list, tuple or a range() which implies the loop number.

Example 1:

```
msg = "Python"
for i in msg:
    print (i)
```

Output:

P
y
t
h
o
n

Example 2:

```
language = ["C","C++","Java","Python"]  
for x in language:  
    print (x)
```

Output:

C
C++
Java
Python

Example 3:

```
words = ['VRS', 'Java', 'Python']  
for w in words:  
    print w, len(w)
```

Output:

VRS 3
Java 4
Python 6

Looping using Sequence of Numbers

- To iterate over a sequence of numbers, the built-in **range()** and **xrange()** function is used over a for loop.
- It is used to indicate how many times the loop will be repeated.
- This function is very useful to generate sequences of numbers in the form of a list.
- The difference between range and xrange is that the range function returns a new list with numbers of that specified range, whereas xrange returns an iterator, which is more efficient.

Syntax

```
range(start, upto, step)
```

- The arguments of range function are used as follows:
 - start and step are both optional.
 - upto must always be there, it means "up to but not including" the value.
 - start, upto, and step must all be integers

Example 1:

```
# One Parameter  
for num in range(5)           #range(stop)  
    print num
```

Output:

0
1
2
3
4

Example 2:

**Write a for loop that prints numbers from 0 to 57 using the range function in python.
[Nov/Dec 2023]**

Two Parameters

```
for num in range(3,6)           #range(start, stop)
    print num
```

Output:

3
4
5

Example 3:**# Three Parameters**

```
for num in range(4,10,2) #range(start, stop, step)
    print num
```

Output:

4
6
8

Examples: of the use of range:

range(10) produces the list: [0,1,2,3,4,5,6,7,8,9]

range(1, 7) produces the list: [1,2,3,4,5,6]

range(0, 30, 5) produces the list: [0,5,10,15,20,25]

range(5, -1, -1) produces the list: [5,4,3,2,1,0]

Definite vs Indefinite Loops / Finite vs Infinite Loops

- A **definite loop** is a loop in which the number of times the loop will iterate can be determined before the loop is executed.
- An **indefinite loop** is a loop in which the number of times that the loop will iterate cannot be determined before the loop is executed.

(Or)

- An **infinite loop** is a loop that executes its block of statements repeatedly until the user forces the program to quit.
- A loop becomes infinite if a condition never becomes FALSE.
- This results in a loop that never ends. Such a loop is called an infinite loop.

CONTROL STATEMENTS

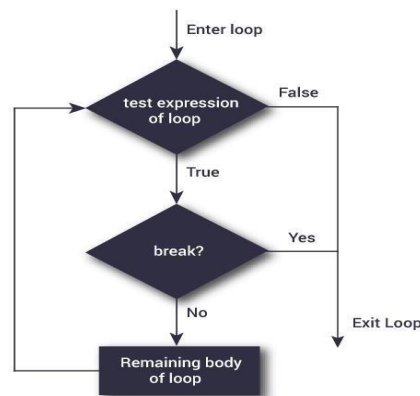
Explain various control statements in python. (Or) Explain with an example break and continue statement in python. (Jan 2018) (Or) Explain the use of continue and break statements with an example. [May 2019] (or) Outline Break and continue statement in Python with an example. [Jan 2022]

break Statement [Jan 2022]. List out the control flow statements in python and explain repetition type in detail with a sample program. [May 2022] *Illustrate the different types of control flow statements in python with flow charts. [Nov/Dec 2023]*

- The break statement is used to terminate or exit a for or a while loop.
- It is used to end the execution of the loop (for or while) immediately and the program control goes to the statement after the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax

break



Example:

```
for letter in 'Python':  
    if letter == 'h':  
        break  
    print (letter)
```

Output:

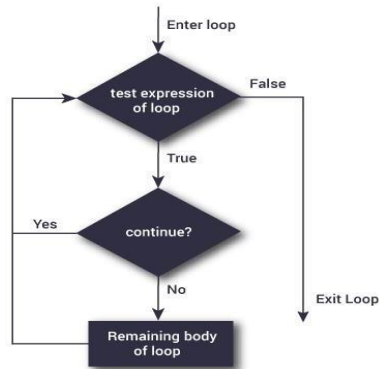
P
y
t

continue Statement [Jan 2022]

- The continue statement is used in a while or for loop to take the control to the beginning of the loop without executing the rest statements inside the loop.

Syntax

continue



Example:

```

for letter in 'Python':
    if letter == 'h':
        continue
    print (letter)
  
```

Output:

```

P
y
t
o
n
  
```

pass Statement

- It is a null operation in python.
- The interpreter reads and executes the pass statement but returns nothing.

Syntax

```

pass
  
```

Example:

```

for letter in 'Python':
    if letter == 'h':
        pass
    print 'This is Pass Block'
print letter
  
```

Output:

```

P
y
t
This is Pass Block
h
o
n
  
```

2 Marks

1. What is Conditional Statement?

- A statement that controls the flow of execution depending on some condition.
- Python provides the following conditional statements or selection structures (decision making).
 - if statement (Conditional)
 - if...else statement (Alternative)
 - if...elif...else statement (Chained Conditional)
 - Nested Conditional

2. Name the two types of iterative statements supported by python. [Jan 2022]

- While loop
- For loop

3. What is while loop?

- The while loop is an entry controlled loop or top tested loop statement
- It means that, the condition is evaluated first, if it is true and then the body of the loop is executed.

4. What is for loop?

- It is used to repeat or execute a series of statements for specific number of times.
- A **stepper variable** is used to count through each iteration of the loop.

5. Define range() Function.

- To iterate over a sequence of numbers, the built-in **range()** and **xrange()** function is used over a for loop.
- It is used to indicate how many times the loop will be repeated.
- This function is very useful to generate sequences of numbers in the form of a list.

6. What is Definite Loop?

- A **definite loop** is a loop in which the number of times the loop will iterate can be determined before the loop is executed.

7. What is Indefinite Loop?

- An **indefinite loop** is a loop in which the number of times that the loop will iterate cannot be determined before the loop is executed.

8. Differentiate between break and continue statement.

What is difference between break and continue in python? Explain with suitable example. [May 2022]

Break	Continue
• It is used to terminate or exit a for or a while loop. • Syntax: break	• It is used in a while or for loop to take the control to the beginning of the loop. • Syntax: continue

9. Do loop statement have else clause? When will it be executed? [Dec 2019]

Yes loop statements have else clause. In most of the programming languages (C/C++, Java, etc), the use of else statement has been restricted with the if conditional statements. But Python also allows us to use the else condition with for loops. The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

Example:

```
for i in range(1, 4):
    print(i)
else: # Executed because no break in for
    print("No Break")
```

Output:

```
1
2
3
No Break
```

FRUITFUL FUNCTIONS

Describe fruitful function in detail. (Or) Why are functions needed? Outline about function definition and call with example. [May 2019] Brief fruitful function and void functions with example. [Dec 2020]

Fruitful Function

- A function that returns a value.

Example:

```
# Creation of Function
function add_f(a, b);
    c = a + b;

    # Returning the value
    return c;
end

# Function call
d = add_f(3, 4);
print(d)
```

Output:

7

FUNCTION

- A named sequence of statements that performs some specific task.
- Functions may or may not take arguments and may or may not produce a result.

Function Definition

- A statement that creates a new function by specifying its name, parameters and the statements it executes.

Syntax

def name of function (list of formal parameters):
body of function

where,

- **def** is a reserved word that tells Python that a function is defined.
- The **function name** is simply a name that is used to refer to the function.

Function Object

- A value created by a function definition.
- The name of the function is a variable that refers to a function object.

Function Call

- A statement that executes a function.
- It consists of the function name followed by an argument list.

Example

```
def max(x, y):           #Function Definition
    if x > y:
        return x
    else:
        return y
print max(3,4)           # Function Call
```

Output

4

Void Function

- A function that doesn't return a value.
- Void functions are those that do not return any value after their calculation is complete.
- These types of functions are used when the calculations of the functions are not needed in the actual program.
- These types of functions perform tasks that are crucial to successfully run the program but do not give a specific value to be used in the program.

Example:

```
# Creation of Function
function add_v(a, b);
c = a + b;
    print(c);
end

# Function Call
add_v(3, 4)
```

Output:

7

- The result of a function.
- If a function call is used as an expression, the return value is the value of the expression which we usually assign to a variable.

```
e = math.exp(1.0)
```

```
height = radius * math.sin(radians)
```

- All of the functions we have written so far are void; they print something, but their return value is None.
- The first example is area, which returns the area of a circle with the given radius:

```
def area(radius):
```

```
    temp = math.pi * radius**2
```

```
    return temp
```

- We have seen the return statement before, but in a fruitful function the return statement includes an expression.
- This statement means: "Return immediately from this function and use the following expression as a return value."
- The expression can be arbitrarily complicated, so we could have written this function more concisely:

```
def area(radius):
```

```
    return math.pi * radius**2
```

- On the other hand, **temporary variables** like temp often make debugging easier.
- Sometimes it is useful to have multiple return statements, one in each branch of a conditional:

```
def absolute_value(x):
```

```
    if x < 0:
```

```
        return -x
```

```
    else:
```

```
        return x
```

- Since these return statements are in an alternative conditional, only one will be executed.
- As soon as a return statement executes, the function terminates without executing any subsequent statements.
- Code that appears after a return statement, or any other place the flow of execution can never reach, is called **dead code**.

Incremental Development

- The goal of incremental development is to avoid long debugging sessions by adding and testing only a small amount of code at a time.
- Consider an example, to find the distance between two points, given by the coordinates x1, y1 and x2, y2.

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- The first step is to consider what a distance function will do. ie., what are the inputs (parameters) and what is the output (return value)?
- In this case, the inputs are two points, which can be represented using four numbers.
- The return value is the distance represented by a floating-point value.
- Function can be written as:

```
def distance(x1, y1, x2, y2):
    return 0.0
```

- Obviously, the above code doesn't compute distances; it always returns zero.
- But it is syntactically correct and it runs, which means that you can test it before you make it more complicated.
- To test the new function, call it with sample arguments:

```
>>> distance(1, 2, 4, 6)
0.0
```

- At this point we have confirmed that the function is syntactically correct, and we can start adding code to the body.
- A reasonable next step is to find the differences $x_2 - x_1$ and $y_2 - y_1$. The next version stores those values in temporary variables and prints them:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    print('dx is', dx)
    print('dy is', dy)
    return 0.0
```

- If the function is working, it should display dx is 3 and dy is 4.
- Next we compute the sum of squares of dx and dy:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    print('dsquared is: ', dsquared)
    return 0.0
```

- Again, you would run the program at this stage and check the output (which should be 25).
- Finally, you can use `math.sqrt` to compute and return the result:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
```

```
result = math.sqrt(dsquared)
return result
```

- If the above code works correctly then, you want to print the value of result before the return statement.
- The final version of the function doesn't display anything when it runs; it only returns a value.
- Code like that is called **scaffolding** because it is helpful for building the program but is not part of the final product.

The key aspects of the process are:

1. Start with a working program and make small incremental changes.
2. Use variables to hold intermediate values so you can display and check them.
3. Once the program is working, you might want to remove some of the scaffolding or consolidate multiple statements into compound expressions, but only if it does not make the program difficult to read.

Parameters

Write short notes on parameters.

- A **parameter** is a variable name that is listed in the parentheses of a function header.
- An **argument** is a value assigned to a function parameter when the function is called.
- Parameters appear in the function definition; arguments appear in the function call.

Flow of Execution

- It is very important to understand what happens when python encounters a function call.
- The rules for executing a function call:
 - Evaluate the arguments to produce memory addresses.
 - Store those memory addresses in the corresponding parameters.
 - Execute the body of the function.
 - When complete, return to the location in the program where the function was called.

Example

```
def Calculate(x,y): # Parameters or Formal Parameters
    z = x + y
    return z
print Calculate(5,6)          # Arguments or Actual Parameters
```

Local and Global Scope (Scope of the Variable)

Discuss in detail about scope of the variable in python.

Comment with an example on the use of local and global variable with the same identifier name.[May 2019]. What is recursion? Write and explain a python program to find factorial of number using recursion.[May 2022]

- Scope of the variable implies the availability of variables within the program.
- Variables have 2 types of scope.
 - Local variables
 - Global variables

Local Variables

- A variable defined inside a function. A local variable can only be used inside its function.
- Local variables have some very desirable properties:
 - The memory required to store a local variable is used only when the variable is in scope.
 - The same variable name can be used in different functions without any conflict.

Global Variables

- A global variable is defined outside of all functions and is not local to any particular function.
- Any function can legally access and/or modify a global variable.

Example 1:

```
def vrs():
    # Local Scope
    s = "Welcome to VRS"
    print s
# Global Scope
s = "Python Programming"
vrs()
print s
```

Output

```
Welcome to VRS
Python Programming
```

Example 2:

```
m = 30 # global variable
def f1(x):
    n = 2 # local variable
    x = n * x
    print(x)
f1(m)
```

Output

```
60
```

Function Composition

Write short notes on function composition.

- The ability of calling one function from another is called function composition.

(Or)

- It is a way of combining functions such that the result of each function is passed as the argument of next function.

Example

- Consider a function that takes two points, the center of the circle and a point on the perimeter, and computes the area of the circle.
- Assume that the center point is stored in the variables xc and yc, and the perimeter point is in xp and yp.
- The first step is to find the radius of the circle, which is the distance between the two points.

For example,

```
radius = distance(xc, yc, xp, yp)
```

- The next step is to find the area of a circle with that radius; For example,
result = area(radius)

- Encapsulating these steps in a function, we get:

```
def circle_area(xc, yc, xp, yp):
    radius = distance(xc, yc, xp, yp)
    result = area(radius)
    return result
```

- The temporary variables radius and result are useful for development and debugging, but once the program is working, we can make it more concise by composing the function calls:

```
def circle_area(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

Compare return value and composition. [April/May-2023-2marks]

Return value:	Compositions:
A "return value" is the value that a function produces as output when it is called and executed.	The ability of calling one function from another is called function composition.
Example: <pre>def add(a, b): return a + b result = add(3, 4) print(result)</pre>	<pre>def square(x): return x * x def add(a, b): return a + b def square_of_sum(x, y): s = add(x, y) return square(s) result = square_of_sum(3, 4) print(result)</pre>

Recursion:

Briefly explain about recursion.

- A function that calls itself again and again is recursive; the process of executing it is called recursion.

Example 1:

```
n=5
def countdown(n):
    if n <= 0:
        print('Python')
    else:
        print(n)
    countdown(n-1)
```

- If n is 0 or negative, it outputs the word, “Python”.
- Otherwise, it outputs n and then calls a function named countdown—itsself—passing n-1 as an argument.

Example 2 (Factorial Program):

```
n=6
def factorial(n):
    if(n <= 1):
        return 1
    else:
        return(n*factorial(n-1))
#n = int(input("Enter number:"))
print "Factorial of",n,"is:",factorial(n)
```

Output:

Factorial of 6 is: 720

Infinite Recursion

- If a recursion never reaches a base case, it goes on making recursive calls forever, and the program never terminates. This is known as **infinite recursion**.

Advantages

- It makes the code clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

2 Marks

1. What is Fruitful Function? [Nov/Dec 2023]

- A function that returns a value.

2. What is Incremental Development?

- The goal of incremental development is to avoid long debugging sessions by adding and testing only a small amount of code at a time.

3. Define Dead Code.

- Code that appears after a return statement or any other place the flow of execution can never reach, is called **dead code**.

4. What is Scaffolding? (Or) Define Scaffolding.

- Code that is helpful for building the program but is not part of the final product, called as **scaffolding**.

5. Define Parameter.

- A **parameter** is a variable name that is listed in the parentheses of a function header.
- Parameters appear in the function definition.

6. Define Argument.

- An **argument** is a value assigned to a function parameter when the function is called.
- Arguments appear in the function call.

7. What is a scope of variable?

- Scope of the variable implies the availability of variables within the program.
- Variables have 2 types of scope.
 - Local variables
 - Global variables

8. Define Local Variable.

- A variable defined inside a function. A local variable can only be used inside its function.

9. Define Global Variable.

- A global variable is defined outside of all functions and is not local to any particular function.
- Any function can legally access and/or modify a global variable.

10. Define Function Composition? (Or) What is Function Composition?

- The ability of calling one function from another is called function composition.
(Or)
- It is a way of combining functions such that the result of each function is passed as the argument of next function.

11. Define Recursion. [Jan 2019] (Or) Define recursive function.[May 2019][Jan 2022]

- A function that calls itself again and again is recursive; the process of executing it is called recursion.

12. Define Infinite Recursion. [Jan 2019]

- Recursion goes on making recursive calls forever and the program never terminates. This is known as **infinite recursion**.

STRINGS

Explain in detail about strings with an example. [Jan 2019] (Or) Analyze string slicing. Illustrate how it is done in python with example. [May 2019]

- A string in Python is a sequence of characters.
- Strings are immutable, which means they cannot be changed after they are created.

String Slices

- It is used to extract any substring from a **string**. **(Or)** A segment of a string is called a **slice**.
- To access substrings, square brackets [] is used for slicing along with the index or indices to obtain your substring.

Hello
0 1 2 3 4
-5 -4 -3 -2 -1

- s[1:4] is 'ell' -- character starting at index 1 and extending up to 3, but not including index 4.
- s[1:] is 'ello' -- returns the character except at index 1 and up to end of the string.
- S[:1] is 'H' -- returns the character at index 1.
- s[:] is 'Hello' -- returns a copy of the whole thing.
- s[-1:] is 'o' -- returns the character at index -1 from the end of the string.
- s[:-1] is 'Hell' -- returns the character except at index -1 from the end of the string.

Index

- An integer value used to select an item in a sequence, such as a character in a string.
- In Python indices start from 0.

Example 1:

```
>>> s = 'Monty Python'
>>> s[0:5]
'Monty'
>>> s[6:12]
'Python'
```

Example 2:

```
a="String"
print (a[1:3])
print (a[:-1])
```

Output:

tr
Strin

Accessing Characters in a String

- We can access individual characters using indexing and a range of characters using slicing.
- Trying to access a character out of index range will raise an `IndexError`.
- The index must be an integer. In Python indices start from 0.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

Example

```
>>> s="Python Programming"
>>> s[:]
'Python Programming'
>>> s[0]
'P'
>>> s[17]
'g'
>>> s[-1]
'g'
>>> s[20]
IndexError: string index out of range
```

Updating Strings

- We can "update" an existing string by (re)assigning a variable to another string.
- The new value can be related to its previous value or to a completely different string altogether.

Traversal with a for Loop

- A lot of computations involve processing a string one character at a time.
- Often they start at the beginning, select each character in turn, do something to it, and continue until the end.
- This pattern of processing is called a **traversal**.

Example

```
str = 'Python'
for letter in str:
    print(letter)
```

Output

P
y
t
h
o
n

Immutability [April/May-2023-2marks]

Python strings are immutable. Justify with an example. [Jan 2019] [Nov/Dec 2023]

- Strings in python are immutable.
- The property of a sequence whose items cannot be changed.
- ie., once string is created it can't be modified.

Example

```
>>> str1="Monty Python"
>>> str2="Monty Python"
```

- Here str1 and str2 refers to the same string object "Monty Python" which is stored somewhere in memory.
- We can test whether str1 refers to same object as str2 using **id()** function.
- **id()** function is used to return the "identity" of an object (memory address).

Example

```
>>> id(str1)
31922800
>>> id(str2)
32806656
>>> id("Monty Python")
32806656
```

- As both str1 and str2 points to same memory location, hence they both points to the same object.

String Functions and Methods

Explain various string methods and functions in detail.

What is string function in python? Explain any three python string methods with an example. [May 2022]

- A string method is like a function, but it runs "on" an object.
- Here are some of the most common string methods: They are,

len()

- It returns the length of the string.

Example

```
>>> s="vrscet"
>>> len(s)
6
```

count()

- It counts the number of occurrences of a character in string.

Example

```
>>> s="Welcome to vrscet"
>>> s.count('e')
```

lower(), upper()

- It returns the lowercase or uppercase letters of the string.

Example

```
>>> s="VrsceT"
>>> s.lower()
'vrscet'
>>> s.upper()
'VRSCET'
```

strip()

- It returns a copy of the string by removing the leading/trailing white space.

Example

```
>>> s=" vrs "
>>> s.strip()
'vrs'
```

lstrip(), rstrip()

- It returns a copy of the string with the leading (lstrip) and trailing (rstrip) with whitespace removed.

Example

```
>>> s=" vrs "
>>> s.lstrip()
'vrs '
>>> s.rstrip()
' vrs'
```

find()

- It returns the position of character 'x' or -1 if not found.

Example

```
>>> s1="Welcome to VRS"
>>> s1.find('e')
1
>>> s1.find('m')
5
>>> s1.find('a')
-1
```

replace()

- It returns a string where all occurrences of 'old' have been replaced by 'new'.
- Syntax: replace('old', 'new')

Example

```
>>> s1="Welcome to VRS"
>>> s1.replace('VRS','VRSCET')
'Welcome to VRSCET'
```

split()

- It returns a list of words separated by the given delimiter.

Example

```
>>> s1="montys flying circus"
>>> s1.split()
['montys', 'flying', 'circus']
```

join()

- It joins or concatenates the elements in the given list together using the string as the delimiter.

Example

```
>>> s=""
>>> list=("vrs","cet")
>>> s.join(list)
'vrscet'
```

isalnum()

- It returns True, if the string consists of alphanumeric characters, otherwise False.

Example

```
>>> s1="vrs123"
>>> s1.isalnum()
True
>>> s2="12345"
>>> s2.isalnum()
True
>>> s2="@# $"
>>> s2.isalnum()
False
```

isalpha()

- It returns True, if the string contains only alphabetic character(s), otherwise False.

Example

```
>>> s="Welcome to vrscet"
s.isalpha()
False
>>> s1="VRSCET"
>>> s1.isalpha()
True
```

isdigit()

- It returns True, if the string contains only digit(s), otherwise False.

Example

```
>>> s="12345"
>>> s.isdigit()
True
>>> s1="vrs123"
```

```
>>> s1.isdigit()
```

```
False
```

isspace()

- It returns True, if the string contains only white space(s), otherwise False.

Example

```
>>> s="Welcome to vrsct"
```

```
>>> s.isspace()
```

```
False
```

```
>>> s1=" "
```

```
>>> s1.isspace()
```

```
True
```

istitle()

- It returns True, if the string is a titlecased, otherwise False.

Example

```
>>> s="welcome to vrs"
```

```
>>> s.istitle()
```

```
False
```

```
>>> s1="Welcome to Vrs"
```

```
>>> s1.istitle()
```

```
False
```

```
>>> s2="Welcome To Vrs"
```

```
>>> s2.istitle()
```

```
True
```

```
>>> s.title()
```

```
'Welcome To Vrs'
```

capitalize()

- It is used to capitalize the first letter of string.

Example

```
>>> s="python programming"
```

```
>>> s.capitalize()
```

```
'Python programming'
```

swapcase()

- It returns a copy of the string in which all the characters are swapped based on their case.

Example

```
>>> s="Welcome to VRS"
```

```
>>> s.swapcase()
```

```
'wELCOME TO vrs'
```

String Module

Explain string module. [April/May 2023]

- This module contains a number of functions to process standard Python string operations.

- Here, most functions are available as string methods.

Example

```
import string
text = "Monty Python's Flying Circus"
print "Uppercase:", string.upper(text)
print "Lowercase:", string.lower(text)
print "Split the String:", string.split(text)
print "Joining the String:", string.join(string.split(text), ",")
print "Replacing the String:", string.replace(text, "Python", "Java")
print "Finding the Character:", string.find(text, "Python"), string.find(text,
"Java")
print "Count the Character:", string.count(text, "n")
```

Output:

```
Uppercase: MONTY PYTHON'S FLYING CIRCUS
Lowercase: monty python's flying circus
Split the String: ['Monty', "Python's", 'Flying', 'Circus']
Joining the String: Monty+Python's+Flying+Circus
Replacing the String: Monty Java's Flying Circus
Finding the Character: 6 -1
Count the Character: 3
```

USING STRING METHODS INSTEAD OF STRING MODULE FUNCTIONS

Example

```
text = "Monty Python's Flying Circus"
print "Uppercase:", text.upper()
print "Lowercase:", text.lower()
print "Split the String:", text.split()
print "Joining the String:", text.join(string.split(text), ",")
print "Replacing the String:", string.replace(text, "Python", "Java")
print "Finding the Character:", string.find(text, "Python"), string.find(text, "Java")
print "Count the Character:", string.count(text, "n")
```

Output

```
Uppercase: MONTY PYTHON'S FLYING CIRCUS
Lowercase: monty python's flying circus
Split the String: ['Monty', "Python's", 'Flying', 'Circus']
Joining the String: Monty+Python's+Flying+Circus
Replacing the String: Monty Java's Flying Circus
Finding the Character: 6 -1
Count the Character: 3
```

The in Operator

- The word **in** is a boolean operator that takes two strings and returns True if the first appears as a substring in the second.

Example 1

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

Example 2

```
def in_both(word1, word2):
    for letter in word1:
        if letter in word2:
            print(letter)

in_both('apples', 'oranges')
```

Output

```
a
e
s
```

Searching a String:

Write a python code to search a string in the given list. [May 2019]

Search function with parameter list name

```
def search(list,n):
    for i in range(len(list)):
        if list[i] == n:
            return True
    return False

# list which contains both string and numbers.
list = [1, 2, 'sachin', 4,'Geeks', 6]
n = 'Geeks'
if search(list, n):
    print("Found")
else:
    print("Not Found")
```

Output:

```
Found
```

LISTS AS ARRAYS

Compare lists and array with example. Can list be considered as an array?

Justify. [Dec 2019]

- Arrays look like a list.
- Arrays and lists in python are both used to store data.
- As a list can contain any type of data there is no need to create a special two-dimensional array.

- Instead we can store lists within lists.

Example

```
>>>myArray=[[1,2],[3,4]]
>>>myArray[0]
[1,2]
>>>myArray[0][1]
[1,2][1] ie., 2
```

- In Python, a 2x2 array is `[[1,2],[3,4]]` with the list `[1,2]` representing the first row and the list `[3,4]` representing the second row.

2 Marks

1. Define String. (Or) What is String?

- A string in Python is a sequence of characters.
- Strings are immutable, which means they cannot be changed after they are created.

2. Define string slice. Give an Example. [Dec 2020]

- It is used to extract any substring from a **string**. (Or) A segment of a string is called a **slice**.
- To access substrings, square brackets `[ ]` is used for slicing along with the index or indices to obtain your substring.

```

Hello
 0  1  2  3  4
-5 -4 -3 -2 -1
```

3. What is indexing in string?

- An integer value used to select an item in a sequence, such as a character in a string.
- In Python indices start from 0.

4. What is Traversal in string?

- A lot of computations involve processing a string one character at a time.
- This pattern of processing is called a **traversal**.

5. Why Python strings are immutable?

- Performance
- Fixed and unchanging storage requirements
- Allows the implementation to safely reuse string objects

6. What are the various string functions and methods?

- A string method is like a function, but it runs "on" an object.
- Here are some of the most common string methods: They are,
 - `len()`
 - `count()`
 - `lower()`, `upper()`
 - `strip()`
 - `find()`
 - `replace()`

7. What is string module?

- This module contains a number of functions to process standard Python string operations.
- Here, most functions are available as string methods.
 - len(), count()
 - lower(), upper()
 - strip(), find(), replace()

8. Write a program to display a set of strings using range() function. [Dec 2019]

```
xyz = ['Janeway', 'Picard', 'Sisko']
for i in range(len(xyz)):
    print(xyz [i])
```

Output:

```
Janeway
Picard
Sisko
```

ILLUSTRATIVE PROGRAMS:

Greatest of Two Numbers

Write a python program to accept two numbers find the greatest and print the result. (Jan 2018)

```
a=10
b=5
if a>b:
    print "a is greater than b"
else:
    print "a is less than b"
```

Output:

```
a is greater than b
```

Factorial

Write a python program to find the factorial of a given number without recursion and with recursion. (Jan 2018)

Without Recursion

```
n=int(input("Enter number:"))
fact=1
while(n>0):
    fact=fact*n
    n=n-1
print("Factorial of the number is: ")
print(fact)
```

Output:

```
Enter number: 5
Factorial of the number is: 120
```

With Recursion

```
n=6
def factorial(n):
    if(n <= 1):
        return 1
    else:
        return(n*factorial(n-1))
```

```
#n = int(input("Enter number:"))
print "Factorial of",n,"is:",factorial(n)
```

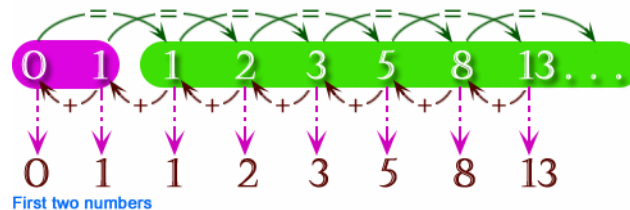
Output:

Factorial of 6 is: 720

Fibonacci Series

Write a python program to generate first N Fibonacci series. (Jan 2018)

Fibonacci Sequence :



Without Recursion

```
n = int(input("Enter the No. of Terms:"))
f1,f2 = 0, 1
f3 = f1 + f2
print("Fibonacci Series of first ", n, " terms")
print(f1)
print(f2)
for i in range (3, n+1)
print(f3)
f1 = f2
f2 = f3
f3 = f1 + f2
```

Output:

Enter the No. of Terms: 5
Fibonacci Series of first 5 terms: 0 1 1 2 3

With Recursion

```
def fibonacci(n):
    if(n <= 1):
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(n):
    print fibonacci(i),
```

Output:

Enter number of terms: 5
Fibonacci sequence:
0 1 1 2 3

Square Root

Without using sqrt() Function

```
num = float(input("Enter a Number: "))
sqrt = num ** 0.5
print "Square Root is:", sqrt
```

Output:

```
Enter a Number: 49
Square Root is 7.0
```

(Or)

Using sqrt() Function

```
import math
num = float(input('Enter a Number: '))
s=math.sqrt(num)
print "Square Root is:", s
```

Output:

```
Enter a Number: 25
The square root is 5.0
```

GCD

```
def gcd(a,b):
    if(b==0):
        return a
    else:
        return gcd(b,a%b)
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
GCD=gcd(a,b)
print "GCD of",a,"and",b,"is:",GCD
```

Output:

```
Enter First Number: 42
Enter Second Number: 28
GCD of 42 and 28 is: 14
```

Exponentiation

```
def pow(x,y):
    if y==0:
        return 1
    res=pow(x,y//2)
    if y%2==0:
        return res*res
    else:
        return x*res*res
```

```
x=int(input("Enter the Value of x:"))
y=int(input("Enter the Value of y:"))
print "The Power of",x,y,"is: ",pow(x,y)
```

Output:

```
Enter the Value of x: 3
Enter the Value of y: 2
The Power of 3, 2 is: 9
```

Sum an Array of Numbers

Using Built-in Method

```
array = [6,7,29]
print sum(array)
```

Output:

```
42
```

(Or)

Without using Function

```
array = [6,7,29]
sum = 0
for i in array:
    sum += i
print sum
```

Output:

```
42
```

(Or)

Using Function

```
def listsum(numList):
    Sum = 0
    for i in numList:
        Sum = Sum + i
    return Sum
print(listsum([1,3,5,7,9]))
```

Output:

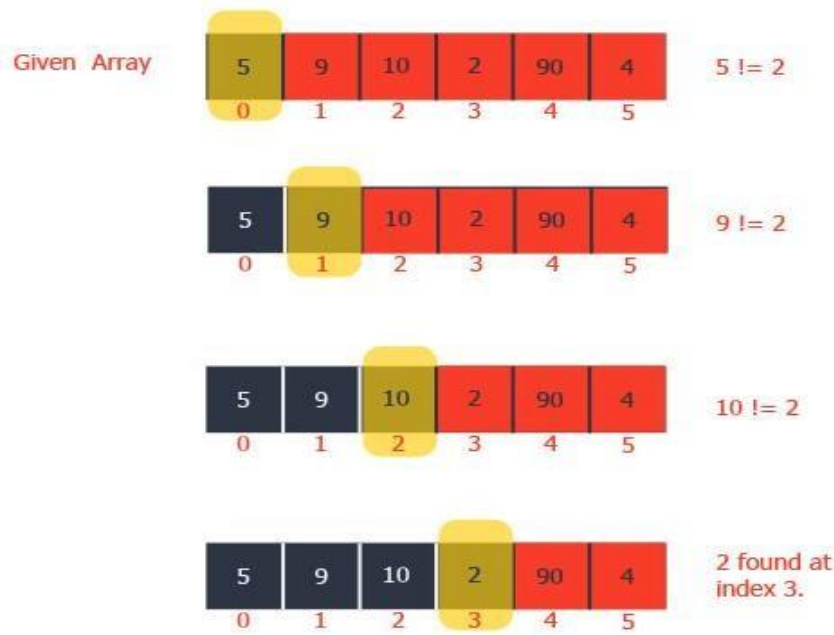
```
25
```

Linear Search

Write a python program to perform linear search on a list. (Jan 2018)

Given an array of n elements, write a function to search a given element x in the array. [Dec 2020]. What is linear search? [May 2022]. (or) Define python lists. How to add elements to the list? Explain with a suitable example program [May2022]. Write a program to perform search operation which sequentially checks each element of the list until a match is found or the whole list has been searched.[April/May-2023]-16marks’.

Linear Search for "2" in 6 elements array



```
size=int(input("Enter the number of Elements:"))
list=[]
flag=False
print("Enter the Elements:")
for i in range(0, size):
    list.append(int(input()))
print "The Elements in the list are:",list
ele=int(input("Enter the Element to be Searched:"))
for i in range(0, size):
    if ele==list[i]:
        flag=True
        break
if flag==True:
    print("The Element {0} was found at position {1}".format(ele,i+1))
elif flag==False:
    print("The Element {} is not found in the List".format(ele))
```

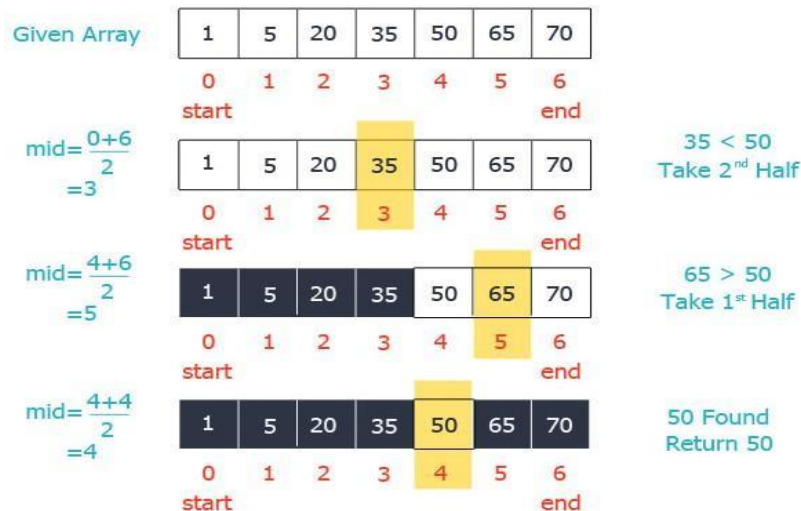
Output:

```
Enter the number of Elements: 5
Enter the Elements: 2 10 4 7 5
The Elements in the list are: [2, 10, 4, 7, 5]
Enter the Element to be searched: 4
The Element 4 was found at position 3
```

Binary Search

Write a python to perform binary search. Trace it with an example of your choice.[Jan 2019] [Nov/Dec 2023]

Binary Search for 50 in 7 elements Array



```
n=int(input("Enter the Number of Elements:"))
num_list=[]
print("Enter the Elements:")
for i in range(0,n):
    num_list.append(int(input()))
print("The Elements in the list are: ", num_list)
ele=int(input("Enter the Element to be Searched:"))
low=0
high=n-1
mid=(low+high)//2
while low<=high:
    if num_list[mid]==ele:
        print "Element:",ele, "is found at position",mid+1
        break
    elif num_list[mid]<ele:
        low=mid+1
    else:
        high=mid-1
    mid=(low+high)//2
if low>high:
    print "Element:",ele," was not found!"
```

Output:

```
Enter the Number of Elements: 5
Enter the Elements: 2 10 4 7 5
```

The Elements in the list are: [2, 10, 4, 7, 5]

Enter the Element to be searched: 10

Element 10 is found at position 2

Write recursive implementation of binary search. [Dec 2020]

Python3 Program for recursive binary search.

Returns index of x in arr if present, else -1

def binarySearch (arr, l, r, x):

 # Check base case

 if r >= l:

 mid = l + (r - l) // 2

 # If element is present at the middle itself

 if arr[mid] == x:

 return mid

 # If element is smaller than mid, then it

 # can only be present in left subarray

 elif arr[mid] > x:

 return binarySearch(arr, l, mid-1, x)

 # Else the element can only be present

 # in right subarray

 else:

 return binarySearch(arr, mid + 1, r, x)

 else:

 # Element is not present in the array

 return -1

Driver Code

 arr = [2, 3, 4, 10, 40]

 x = 10

Function call

 result = binarySearch(arr, 0, len(arr)-1, x)

 if result != -1:

 print ("Element is present at index % d" % result)

 else:

 print ("Element is not present in array")

Output:

Element is present at index 3

If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, it is clear that you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle: If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a “degenerate” triangle.)
[Dec 2019]

i) Write a function named `is_triangle` that takes three integers as arguments, and that prints either “Yes” or “No,” depending on whether you can or cannot form a triangle from sticks with the given lengths.

```
def is_triangle(side_a, side_b, side_c):
    if (side_a + side_b) < side_c or (side_b + side_c) < side_a or (side_a + side_c) < side_b:
        print "No"
    else:
        print "Yes"
```

ii) Write a function that prompts the user to input three stick lengths, converts them to integers, and uses `is_triangle` to check whether sticks with the given lengths can form a triangle.

```
def get_values_and_check_is_triangle():
    side_a = int(raw_input("What is the length of side a?\n"))
    side_b = int(raw_input("What is the length of side b?\n"))
    side_c = int(raw_input("What is the length of side c?\n"))
    is_triangle(side_a, side_b, side_c)
```

Write a python program to generate all permutations of a given string using built in function. [Dec 2019]

```
def toString(List):
    return ''.join(List)
def permute(a, l, r):
    if l == r:
        print toString(a)
    else:
        for i in xrange(l, r + 1):
            a[l], a[i] = a[i], a[l]
            permute(a, l + 1, r)
            a[l], a[i] = a[i], a[l]
string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n-1)
```

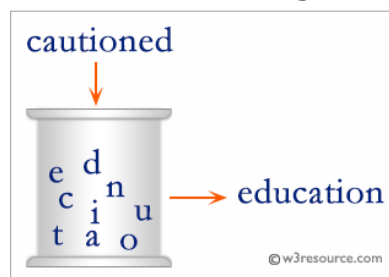
OUTPUT:

ABC
 ACB
 BAC
 BCA
 CBA
 CAB

Write a python function are anagram1 () to check whether two strings are anagram of each other or not with build in string function and are anagram2 ()to check the anagram without build in string function. [Dec 2019]

An anagram is direct word switch or word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once.

For example, the word anagram can be rearranged into nag-a-ram.

**Example 1:**

```

def is_anagram(str1, str2):
    list_str1 = list(str1)
    list_str1.sort()
    list_str2 = list(str2)
    list_str2.sort()
    return (list_str1 == list_str2)
print(is_anagram('anagram','nagaram'))
print(is_anagram('cat','rat'))
  
```

Output:

True

False

Example 2:

```

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

Output:

The strings are anagrams.

There are 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n) [Dec 2020]

Input : ar1[] = {1, 12, 15, 26, 38}

ar2[] = {2, 13, 17, 30, 45}

Output : 16

Explanation :

After merging two arrays, we get

{1, 2, 12, 13, 15, 17, 26, 30, 38, 45}

Middle two elements are 15 and 17

Average of middle elements is $(15 + 17)/2$
which is equal to 16

A Simple Merge based O(n) Python 3 solution

to find median of two sorted lists

This function returns median of ar1[] and ar2[].

Assumptions in this function:

Both ar1[] and ar2[] are sorted arrays

Both have n elements

def getMedian(ar1, ar2 , n):

 i = 0 # Current index of i/p list ar1[]

 j = 0 # Current index of i/p list ar2[]

 m1 = -1

 m2 = -1

 # Since there are 2n elements, median

 # will be average of elements at index

 # n-1 and n in the array obtained after

 # merging ar1 and ar2

 count = 0

 while count < n + 1:

 count += 1

 # Below is to handle case where all

 # elements of ar1[] are smaller than

 # smallest(or first) element of ar2[]

 if i == n:

 m1 = m2

 m2 = ar2[0]

 break

 # Below is to handle case where all

 # elements of ar2[] are smaller than

 # smallest(or first) element of ar1[]

 elif j == n:

 m1 = m2

 m2 = ar1[0]

 break

 if ar1[i] < ar2[j]:

 m1 = m2 # Store the prev median

 m2 = ar1[i]

 i += 1

```

else:
    m1 = m2 # Store the prev median
    m2 = ar2[j]
    j += 1
return (m1 + m2)/2
# Driver code to test above function
ar1 = [1, 12, 15, 26, 38]
ar2 = [2, 13, 17, 30, 45]
n1 = len(ar1)
n2 = len(ar2)
if n1 == n2:
    print("Median is ", getMedian(ar1, ar2, n1))
else:
    print("Doesn't work for arrays of unequal size")

```

Output:

Median is 16.0

What are immutable strings? For the given sentence sort word of sentence in ascending order: Chennai Anna University [Dec 2020]

- A string in Python is a sequence of characters. Strings are immutable, which means they cannot be changed after they are created.

```

sent = "Chennai Anna University"
# Split on space.
my_arr = sent.split(" ")
print(my_arr)
# Create a new array using the sorted method
new_arr = sorted(my_arr)
print(new_arr)

```

Output:

```

['Chennai', 'Anna', 'University']
['Anna', 'Chennai', 'University']

```

Write a python program using function to find the sum of first 'n' odd numbers and print the result. [Jan 2022]

```

def odd(n):
    i=1
    sum=0
    while i<n:
        if i%2==1:
            sum=sum+i
        i=i+1
    print ('Sum of Odd Numbers', sum)
n=int(input('Enter the Limit:'))
odd(n)

```

Output

```

Enter the Limit: 15
Sum of Odd Numbers 49

```

Write a program to find the sum and average of array of numbers.[April/May-2023]-8 Marks

Program:

```
# Get the number of elements in the array from the user
num_elements = int(input("Enter the number of elements in the array: "))

# Initialize an empty list to store the numbers
numbers = []

# Get the numbers from the user and store them in the list
for i in range(num_elements):
    num = float(input(f"Enter element {i + 1}: "))
    numbers.append(num)

# Calculate the sum of the numbers
total_sum = sum(numbers)

# Calculate the average
average = total_sum / num_elements

# Print the sum and average
print(f"Sum of the numbers: {total_sum}")
print(f"Average of the numbers: {average:.2f}")
```

Output:

```
Enter the number of elements in the array: 3
Enter element 1: 12
Enter element 2: 23
Enter element 3: 34
Sum of the numbers: 69.0
Average of the numbers: 23.00
```