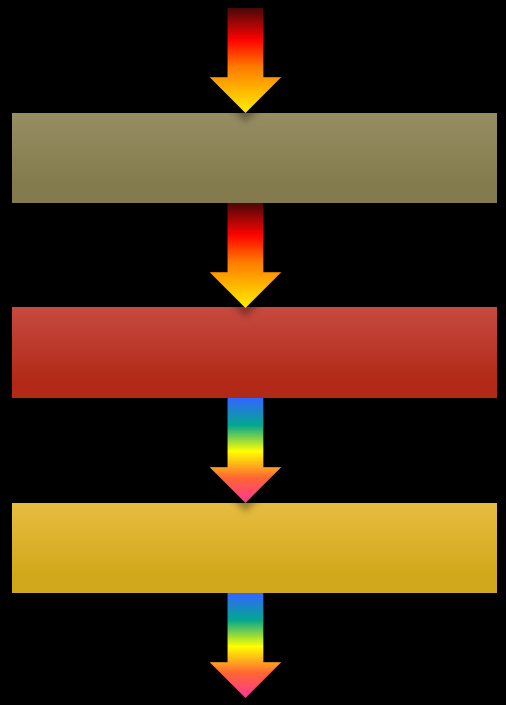
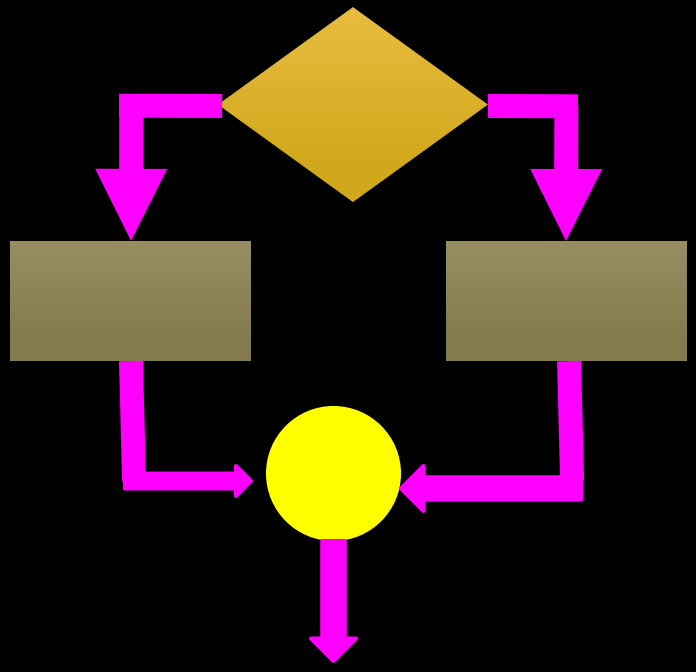
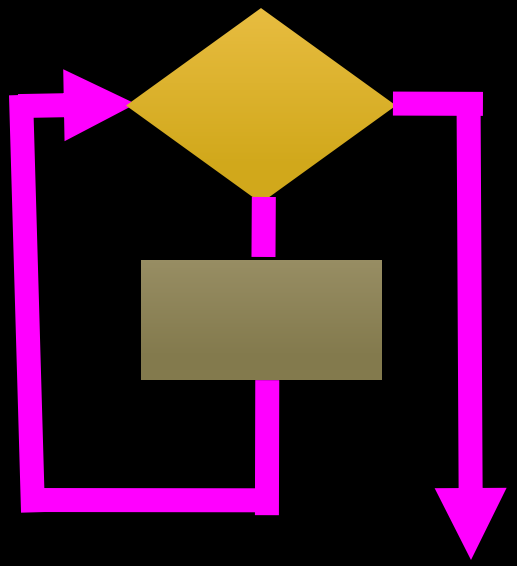
CONDITIONAL AND ITERATIVE STATEMENTS

PROF. SARAH PRITHVIKA P.C.
ASSISTANT PROFESSOR
SCOPE







SEQUENCE SELECTION ITERATION

CONDITIONAL AND ITERATIVE STATEMENTS

INTRODUCTION

INTRODUCTION

Programs are written for the solution to the real world problems. A language should have the ability to control the flow of execution so that at different intervals different statements can be executed. Structured programming is a paradigm aims at controlling the flow of execution of statements in a program by using control structures.

A language which supports the control structures is called as structured programming language

TYPES OF CONTROL STRUCTURES

TYPES OF CONTROL STRUCTURES

A Structured programming is an important feature of a programming language which comprises following logical structure:

1. SEQUENCE
2. SELECTION
3. ITERATION OR LOOPING
4. JUMPING STATEMENTS

1. SEQUENCE

Sequence is the default control structure; instructions are executed one after another.

Statement 1
Statement 2
Statement 3
……..
……..
……..

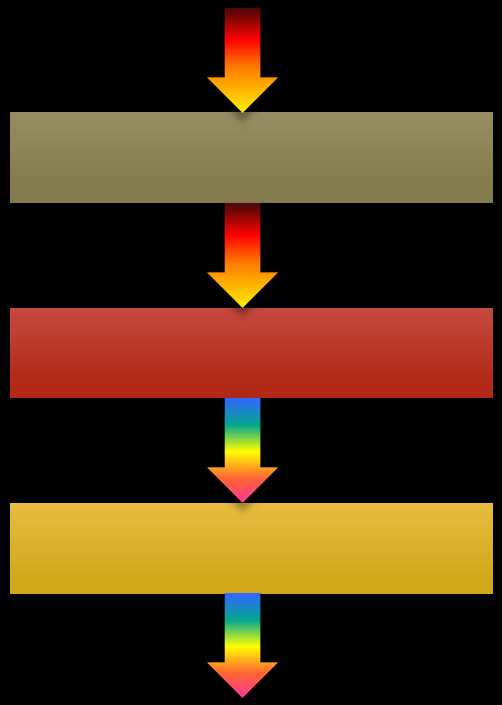
1. SEQUENCE – FLOW CHART

1. SEQUENCE - FLOW CHART

Statement 1

Statement 2

Statement 3



SEQUENCE

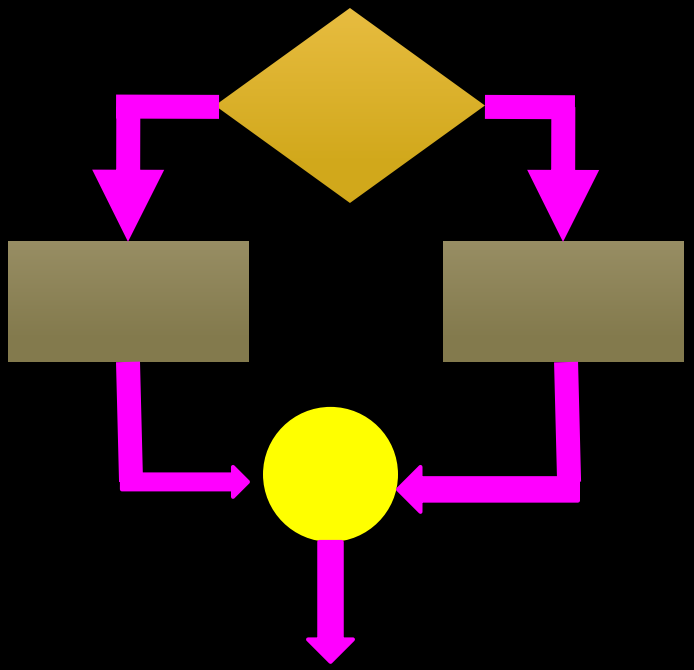
1. SEQUENCE - PROGRAM

1. SEQUENCE - PROGRAM

Sequence is the default control structure; instructions are executed one after another.

```
# This program adds two numbers
num1 = 1.5
num2 = 6.3
sum = float(num1) + float(num2)
print('The sum is =', sum)
```

2. SELECTION



SELECTION

2. SELECTION or CONDITIONAL STATEMENT

A selection statement causes the program control to be transferred to a specific flow based upon whether a certain condition is true or not.

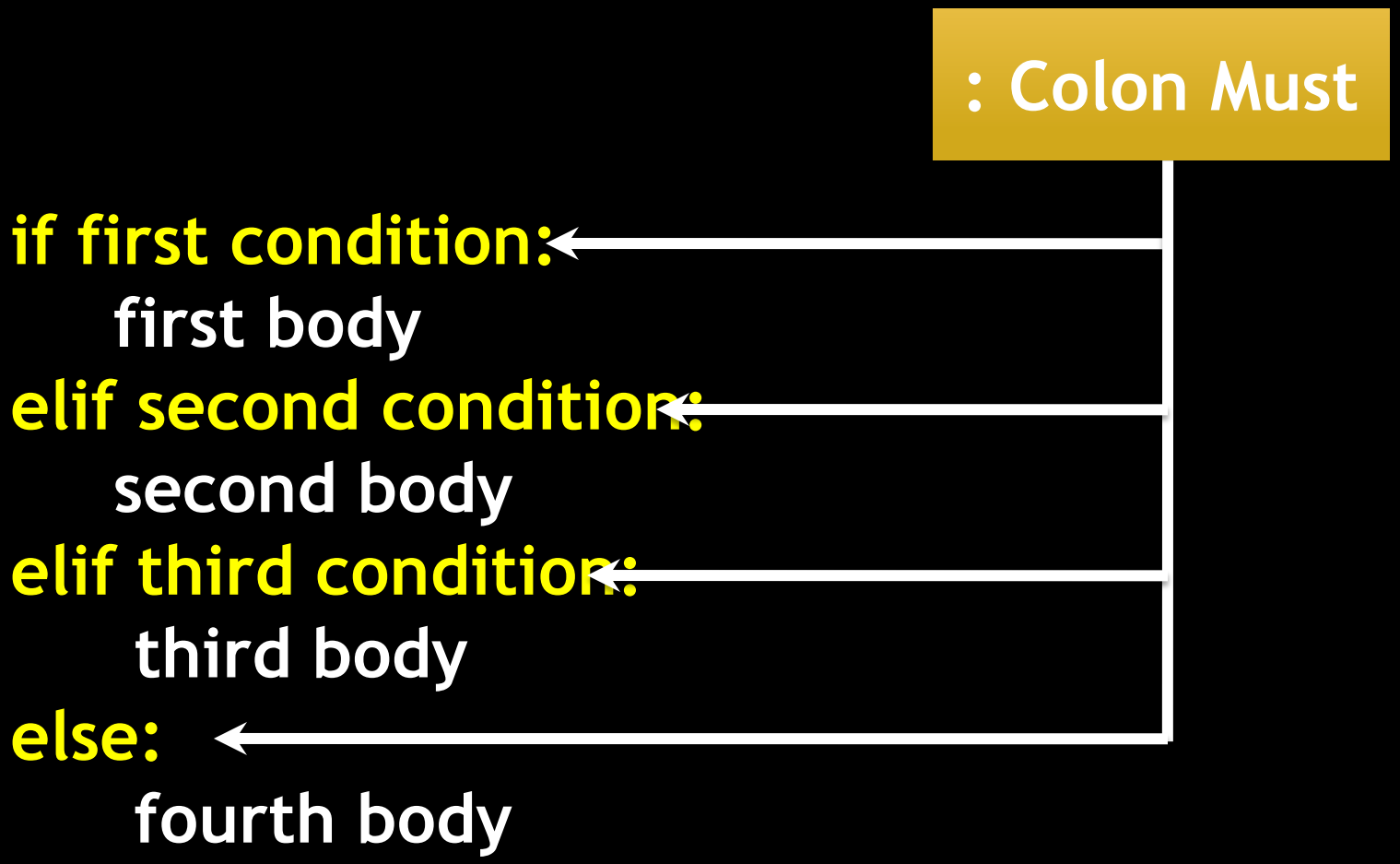
CONDITIONAL CONSTRUCT – if else STATEMENT

CONDITIONAL CONSTRUCT – if else STATEMENT

Conditional constructs (also known as if statements) provide a way to execute a chosen block of code based on the run-time evaluation of one or more Boolean expressions. In Python, the most general form of a conditional is written as follows:

Contd.. Next Slide

CONDITIONAL CONSTRUCT - if else STATEMENT

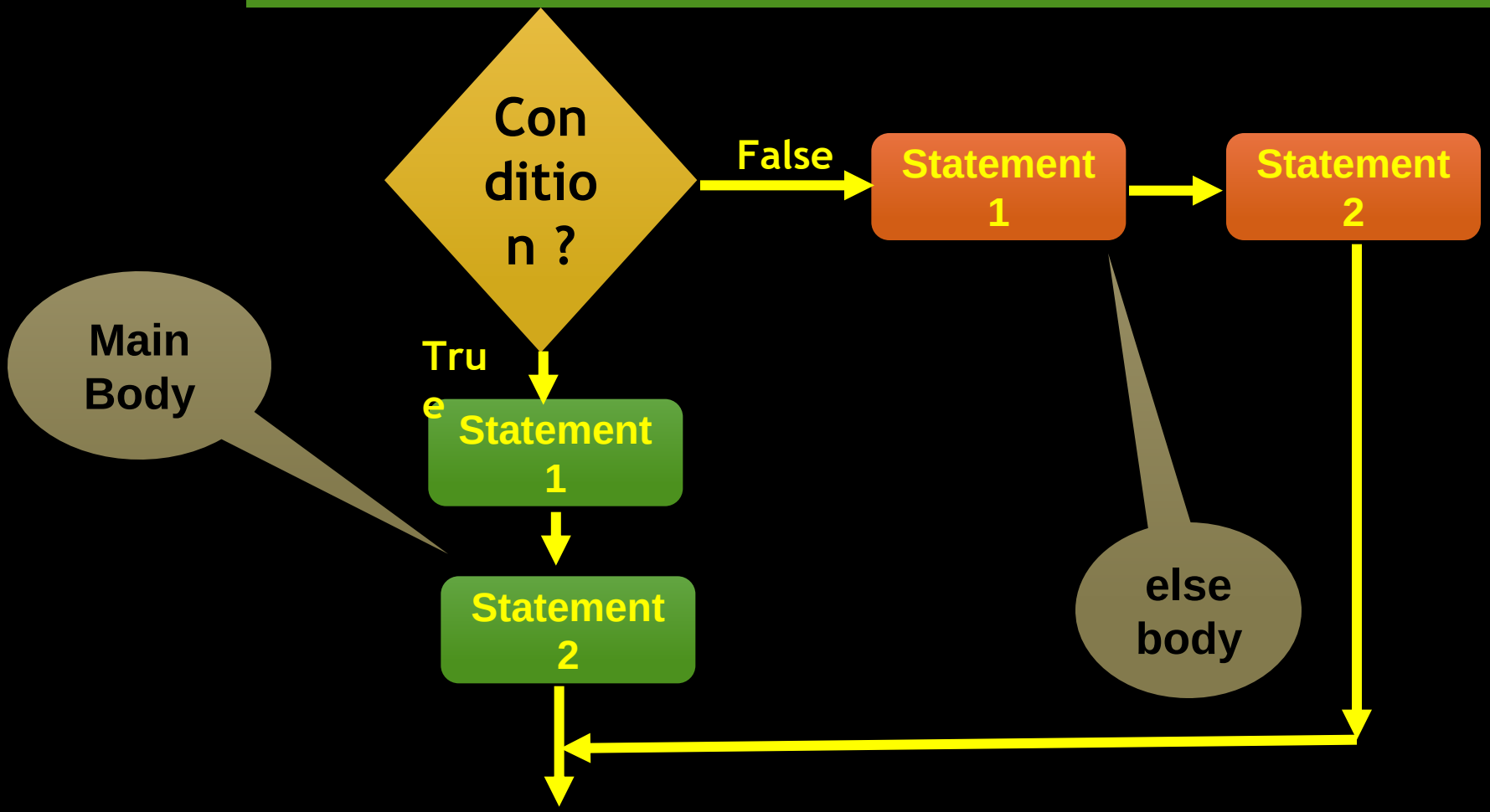


CONDITIONAL CONSTRUCT - if else STATEMENT

FLOW CHART

CONDITIONAL CONSTRUCT – if else STATEMENT

FLOW CHART



CONDITIONAL CONSTRUCT – if else STATEMENT

✔Each condition is a Boolean expression, and each body contains one or more commands that are to be executed conditionally.

✔If the first condition succeeds, the first body will be executed; no other conditions or bodies are evaluated in that case.

CONDITIONAL CONSTRUCT – if else STATEMENT

✔If the first condition fails, then the process continues in similar manner with the evaluation of the second condition. The execution of this overall construct will cause precisely one of the bodies to be executed.

✔There may be any number of elif clauses (including zero), and
✔ The final else clause is optional.

CONDITIONAL CONSTRUCT – if else STATEMENT

EXAMPLE - PROGRAM

EXAMPLES - if STATEMENT

```
a=5
if(a<10):
    print("a is less than 10")
```

else is missing, it is an optional statement

OUTPUT

```
a is less than 10
```

CONDITIONAL CONSTRUCT

EXAMPLE – if else STATEMENT

EXAMPLE - if else STATEMENT

```
age=15
if(age>=18):
    print("Eligible for Voting")
else:
    print("Not Eligible for Voting")
```

: Colon Must

else is used

OUT PUT

```
Not Eligible for Voting
```

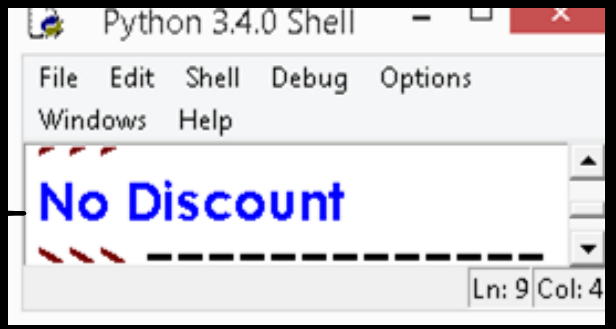
CONDITIONAL CONSTRUCT

EXAMPLES - if elif STATEMENT

EXAMPLES - if elif STATEMENT

```
Age = 27
if Age >= 60:
    print ('Senior Discount')
elif 18 <= Age < 60:
    print ('No Discount')
else:
    print ('Junior Discount')
```

OUTPUT



TERNARY OPERATOR

- The Ternary Operator determines if a condition is true or false and then returns the appropriate value as the result.
- The ternary operator is useful in cases where we need to assign a value to a variable based on a simple condition, and we want to keep our code more concise — all in just one line of code.

```
# Program to demonstrate ternary operator
        a = 10
        b = 20
        # python ternary operator
        print("a is minimum" if a < b else "b is minimum")
```

Nested if

```
c=21
if c<25:
    if c%2==0:
        print("c is an even number less than 25")
    else:
        print("c is an odd number less than 25")
else:
    print("c is greater than 25")
```

```
c is an odd number less than 25
```

Nested if

```
age=33;gender='M'
if age < 18:
    if gender == 'M':
        print('son')
    else:
        print('daughter')
elif age >= 18 and age < 65:
    if gender == 'M':
        print('father')
    else:
            print('mother')
else:
    if gender == 'M':
        print('grandfather')
    else:
        print('grandmother')
```

If and else without elif

Nested if statements	Example use

```
if condition:
    statements
else:
    if condition:
        statements
    else:
        if condition:
            statements

            etc.
```

```
if grade >= 90:
    print('Grade of A')
else:
    if grade >= 80:
        print('Grade of B')
    else:
        if grade >= 70:
            print('Grade of C')
        else:
            if grade >= 60:
                print('Grade of D')
            else:
                print('Grade of F')
```

PROGRAM LIST ON if CONSTRUCT

PROGRAM LIST ON if CONSTRUCT

1. Write a PYTHON program that reads a number and checks if the number is zero or non zero value.
2. Write a PYTHON program to find the largest of two numbers.
3. Write a PYTHON program that reads the number and check the no is positive or negative.
4. Write a PYTHON program to check entered character is vowel or consonant.
5. Write a program to find the largest of three numbers

PROGRAM LIST ON if CONSTRUCT

6. Write a PYTHON program to evaluate the student performance

If percent is >=90 then Excellent performance
If percent is >=80 then Very Good performance
If percent is >=70 then Good performance
If percent is >=60 then average performance
else Poor performance.

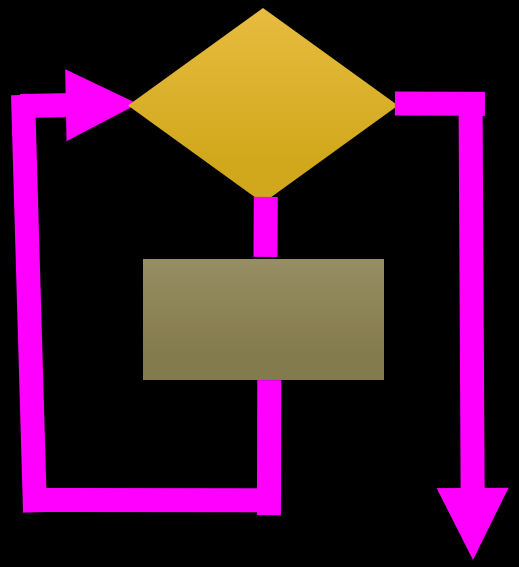
7.Write a PYTHON program to check whether number is even or odd.

8.Write a program to find if the given year is a leap year.

PROGRAM LIST ON if CONSTRUCT

9.Find the roots of a quadratic equation. Use the sqrt function in math module to find the square root

3. ITERATION OR LOOPING



ITERATION

3. ITERATION OR LOOPING

What is loop or iteration?

Loops can execute a block of code number of times until a certain condition is met.

OR

The iteration statement allows instructions to be executed until a certain condition is to be fulfilled.

The iteration statements are also called as loops or Looping statements.

3. ITERATION OR LOOPING

Python provides two kinds of loops & they are,

- while loop
- for loop

while loop

while loop

A while loop allows general repetition based upon the repeated testing of a Boolean condition

The syntax for a while loop in Python is as follows:

```
while condition:
    body
```

: Colon Must

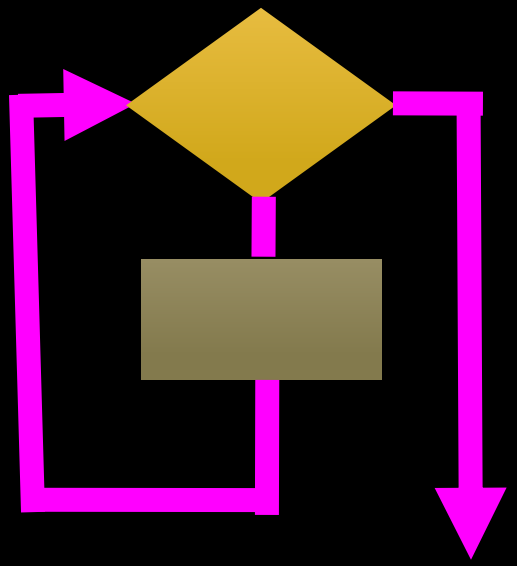
Where, loop body contain the single statement or set of statements (compound statement) or an empty statement.

Contd..

while loop

The loop iterates while the expression evaluates to true, when expression becomes false the loop terminates.

FLOW CHART



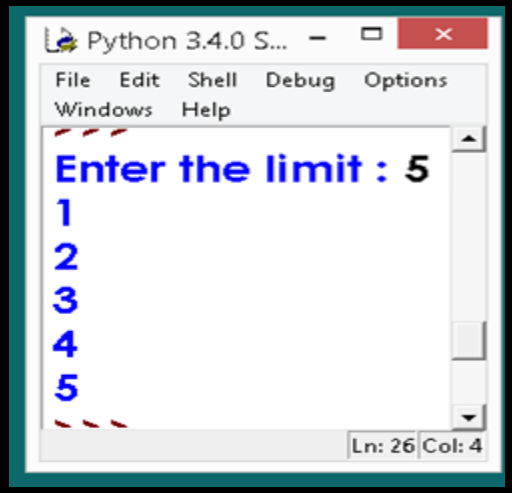
while loop - Programming example

while loop - programs

Natural Numbers generation

```
i=1
n=int(input("Enter the limit : "))
while(i<=n):
    print(i)
    i+=1
```

OUTPUT

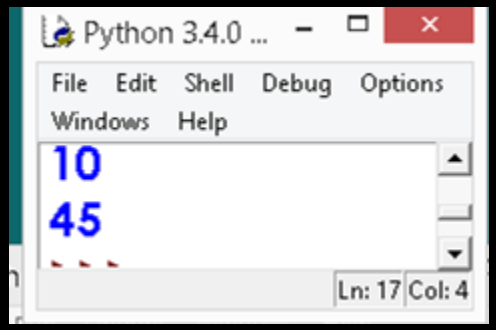


while loop - programs

Calculating Sum of Natural Numbers

```
sum1 = 0
count = 1
while (count < 10):
    sum1 = sum1 + count
    count = count + 1
print (count) # should be 10
print (sum1) # should be 45
```

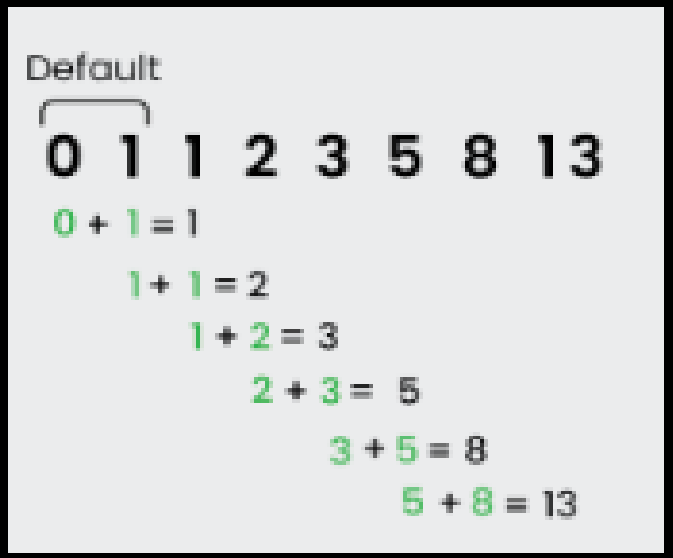
OUTPUT



loop - programs

#Fibonacci numbers

- The Fibonacci sequence is a series where each number is the sum of the two that precede it. It starts from 0 and 1 usually.
- The Fibonacci sequence is given by 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so on.
- The numbers in the Fibonacci sequence are also called Fibonacci numbers.



While loop - programs

#Generating Fibonacci numbers

```
n = 5
a, b = 0, 1

i=0
while i<n:
    print(a, end=" ")
    a, b = b, a + b
    i+=1
```

Output

```
0 1 1 2 3
```

end=‘ ’

The end parameter in the print function is used to add any string at the end of the output of the print statement in python.
By default, the print function ends with a newline.
Passing the whitespace to the end parameter (end=‘ ‘) indicates that the end character has to be identified by whitespace and not a newline.

```
print(“VIT ”)
print(“is awesome”)
```

Output:

```
VIT
is awesome
```

```
print(“VIT ”, end=‘’)
print(“is awesome”)
```

Output:

```
VIT is awesome
```

For example:

```
print(“VIT”, end=’ says ‘)
print(“you are awesome”)
```

Output:

```
VIT says you are awesome
```

break

```
while True:
    user_input = input("Enter a number (or 'quit' to exit): ")

    if user_input.lower() == 'quit':
        print("Exiting the loop.")
        break

    number = float(user_input)
    if number > 0:
        print(f"You entered a positive number: {number}")
    else:
        print(f"You entered a negative number: {number}")
```

```
Enter a number (or 'quit' to exit): 3
You entered a positive number: 3.0
Enter a number (or 'quit' to exit): -5
You entered a negative number: -5.0
Enter a number (or 'quit' to exit): 3
You entered a positive number: 3.0
Enter a number (or 'quit' to exit): -8
You entered a negative number: -8.0
Enter a number (or 'quit' to exit):
```

While/Do While

While Loop

In Python, a while loop repeatedly executes a block of code as long as a given condition is true. It is called as entry controlled loop. Here is an example of a while loop

```
# Example of a while loop
count = 0
while count < 5:
    print("Count is:", count)
    count += 1
```

While/Do While

Do While Loop

Python does not have a built-in do while loop like some other programming languages (e.g., C, C++). However, you can mimic the behavior of a do while loop using a while loop with a condition that is checked at the end of the loop. It is called as exit controlled loop. Here's an example:

```
# Mimicking a do while loop in Python
count = 0
while True:
    print("Count is:", count)
    count += 1
    if count >= 5:
        break
```

While/Do While

While Loop	Do-While Loop
The while loop evaluates the condition first and then execute the statements.	The do-while loop executes the statements first before evaluating the condition.
The condition is specified at the beginning of the loop.	The condition isn't specified until after the body of the loop.
The body is executed only if a certain condition is met and it terminates when the condition is false.	The body is always executed at least once, regardless of whether the condition is met.

while loop - Programs

Class work / Home Work

while loop - programs

1. Write a PYTHON program to print even numbers up to n
2. Write a PYTHON program to print natural numbers up to n in reverse order.
3. Write a PYTHON program to check the entered number is prime or not
4. Write a PYTHON program to print the multiplication table
5. Write a PYTHON program to check the entered number is palindrome or not
6. Print prime numbers in a range. eg 0 to 100

for LOOP

for LOOP

Python's for-loop syntax is a more convenient alternative to a while loop when iterating through a series of elements. The for-loop syntax can be used on any type of iterable structure, such as a list, tuple str, set, dict, or file

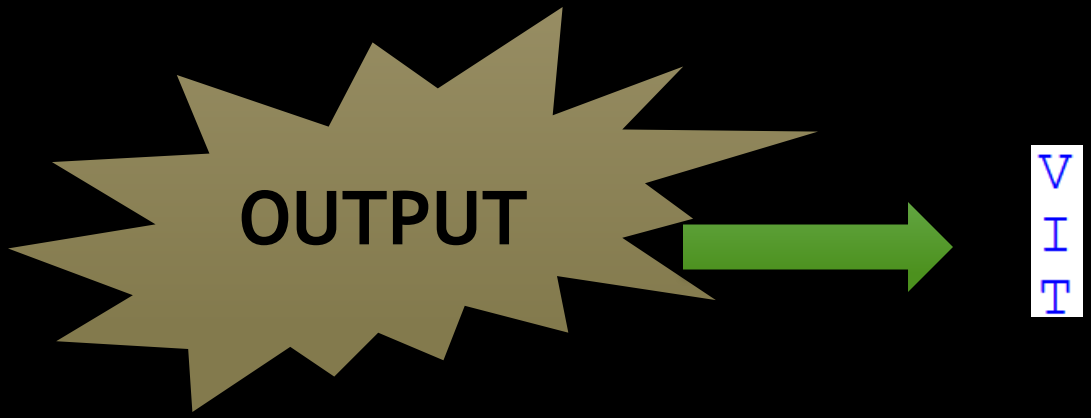
Syntax or general format of for loop is,

```
for element in iterable:
    body
```

for LOOP

Till the list exhaust for loop will continue to execute.

```
name="VIT"
for i in name:
    print(i)
```




```
V
I
T
```

For and Strings

for iterating_var in sequence or range:

statement(s)

Example:

```
for letter in ‘python':
    print ('Current Letter :', letter)
```

Output

```
Current Letter : P
 Current Letter : y
 Current Letter : t
 Current Letter : h
 Current Letter : o
 Current Letter : n
```

for LOOP

range KEYWORD

for LOOP - range KEYWORD

The range() function in Python is a built-in function used to generate a sequence of numbers.

Syntax

```
range(start, stop, step)
```

Parameter Values

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

for LOOP - range KEYWORD

The range() function can be used with one, two, or three arguments:

range(stop)

Generates numbers from 0 up to but not including stop.

Example: range(5) generates numbers 0, 1, 2, 3, 4.

range(start, stop)

Generates numbers from start up to but not including stop.

Example: range(2, 5) generates numbers 2, 3, 4.

range(start, stop, step)

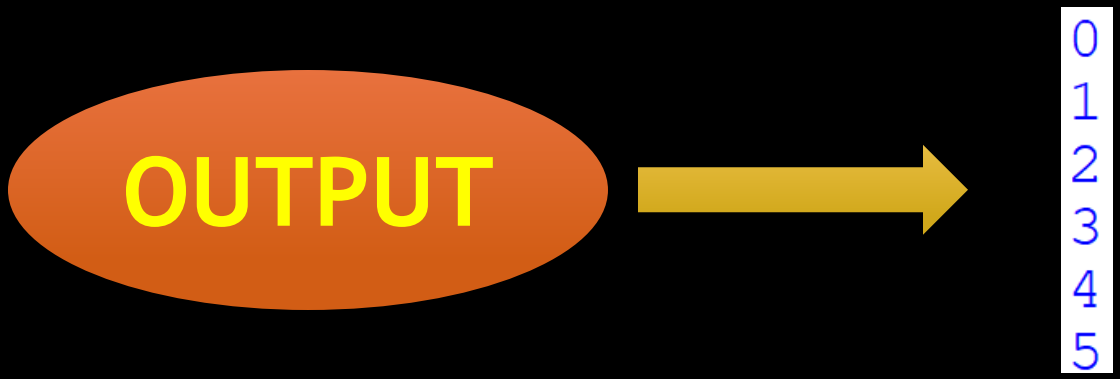
Generates numbers from start up to but not including stop, incrementing by step.

Example: range(2, 10, 2) generates numbers 2, 4, 6, 8.

for LOOP - range KEYWORD

#Generating series of numbers

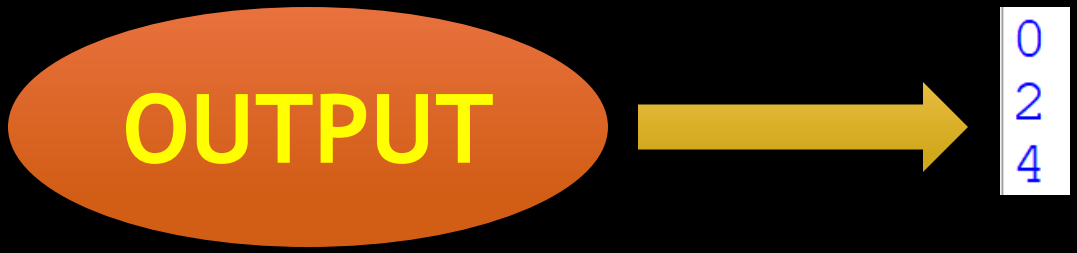
```
name=6
for i in range(0,name,1):
    print(i)
```



for LOOP - range KEYWORD

#Generating even numbers

```
name=6
for i in range(0,name,2):
    print(i)
```

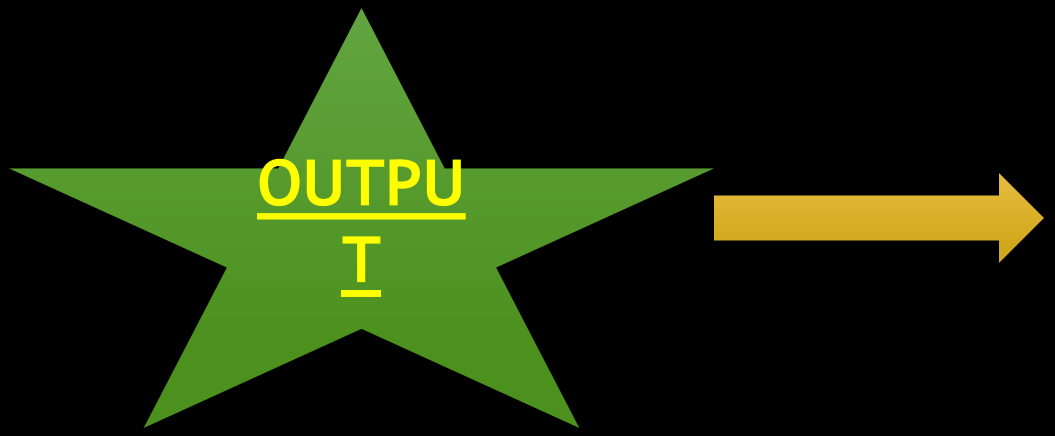


for LOOP - len() FUNCTION

for LOOP - range KEYWORD

print string character by character

```
name=input("Enter string")
for i in range(0,len(name)):
    print(name[i])
```




```
Enter string computer science

c
o
m
p
u
t
e
r

s
c
i
e
n
c
e
```

For loop - programs

#Generating Fibonacci numbers

```
n = 5
a, b = 0, 1
'''
The underscore is a convention meaning
“doesn't matter what this is”.
So the loop will iterate n times,
but without making use of a classic
iteration counter.
'''
for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b
```

Output

```
0 1 1 2 3
```

for loop - Programs

Class work / Home Work

for loop - Programs - Class work / Home Work

1. Write a PYTHON program to print the natural numbers up to n
2. Write a PYTHON program to print even numbers up to n
3. Write a PYTHON program to print odd numbers up to n
4. Write a PYTHON program that prints 1 2 4 8 16 32 … n^2
5. Write a PYTHON program to sum the given sequence

1 + 1/ 1! + 1/ 2! + 1/3! + …. + 1/n!

Note- use the math.factorial function

for vs while

Can all “for loops” be converted to “while loops”?
Yes
Can all “while loops” be converted to “for loops”?
No
while Loop: Suitable for scenarios where the number of iterations is not known in advance and depends on a condition.
eg

```
#Program to find sum of digits
num = int(input("Enter a number"))
n=num
sum = 0
while n > 0:
    sum += n % 10
    n //= 10
print(f'Sum of digits in {num} is {sum}')
```

Calculator program using while

eg

```
while True:
    # Display menu
    print("\nSimple Calculator")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    # Get user choice
    choice = input("Enter choice (1/2/3/4): ")

    # Get numbers from user
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    # Perform operation based on user choice
    if choice == '1':
        result = num1 + num2
        print(f"{num1} + {num2} = {result}")
    elif choice == '2':
        result = num1 - num2
        print(f"{num1} - {num2} = {result}")
    elif choice == '3':
        result = num1 * num2
        print(f"{num1} * {num2} = {result}")
    elif choice == '4':
        if num2 != 0:
            result = num1 / num2
            print(f"{num1} / {num2} = {result}")
        else:
            print("Error! Division by zero.")
    else:
        print("Invalid choice. Please choose a valid operation.")

    # Ask if the user wants to perform another calculation
    continue_calculating = input("\nDo you want to perform another calculation? (yes/no): ").lower()
    if continue_calculating != 'yes':
        print("Exiting the calculator. Goodbye!")
        break
```

for vs while

for Loop: The for loop is advantageous for its simplicity, readability, and ease of use when dealing with sequences, ranges, and iterables.

else statement in loop

else statement in loop

- **else can be used in for and while loops**
- else is executed when the for/while loop completes its execution without interruption by a break statement.

else statement in for loop

```
import math
num = 11
if num > 1:
  for i in range(2, int(math.sqrt(num)+1)):
     if (num % i) == 0:
        print(num, "is not a prime number")
        break
  else:
     print(num, "is a prime number")
```

else statement in while loop

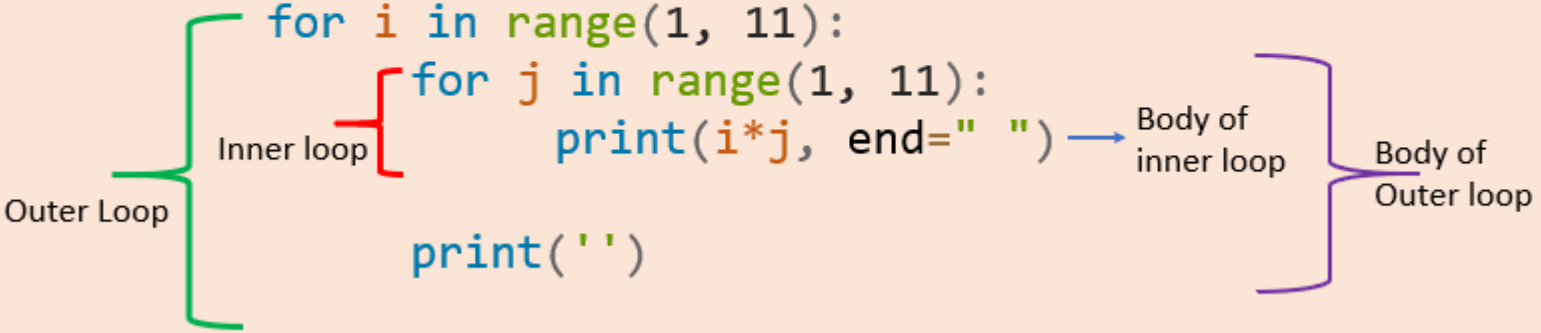
```
import math
num = 12
i=2
if num > 1:
  while i <=int(math.sqrt(num)):
     if (num % i) == 0:
         print(num, "is not a prime number")
         break
     i+=1
  else:
     print(num, "is a prime number")
```

Nested Loop

- A nested loop is a loop inside the body of the outer loop. The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer `for` loop can contain a `while` loop and vice versa.
- The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops.
- In the nested loop, the number of iterations will be equal to the number of iterations in the outer loop multiplied by the iterations in the inner loop.
- In each iteration of the outer loop inner loop execute all its iteration.
- For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.

Nested Loop

Nested For loop



Nested for Loop

```
for i in range(0, 3):
    for j in range(0,3):
        print(i,j)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

Nested Loop

```
i=0
while i <3:
    j=0
    while j <3:
        print(i,j)
        j=j+1
    i=i+1
```

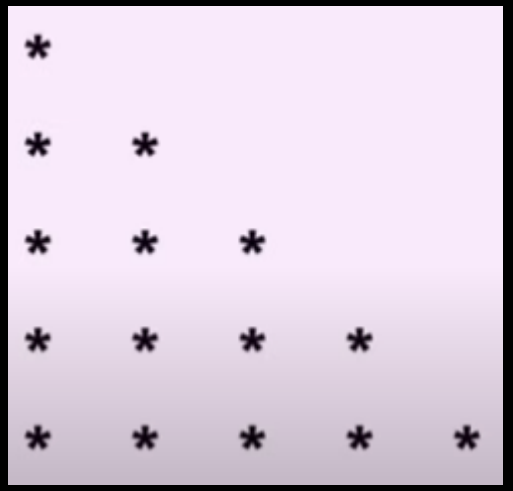
```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

Nested Loop

```
n=5
for i in range(n):
    for j in range(n):
        print("*",end=' ')
    print()
```

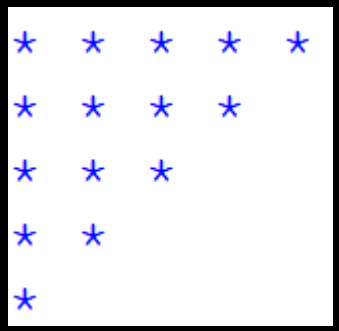
Nested Loop

```
#increasing triangle pattern
n=5
for i in range(n):
    for j in range(i+1):
        print("*",end=' ')
    print()
```



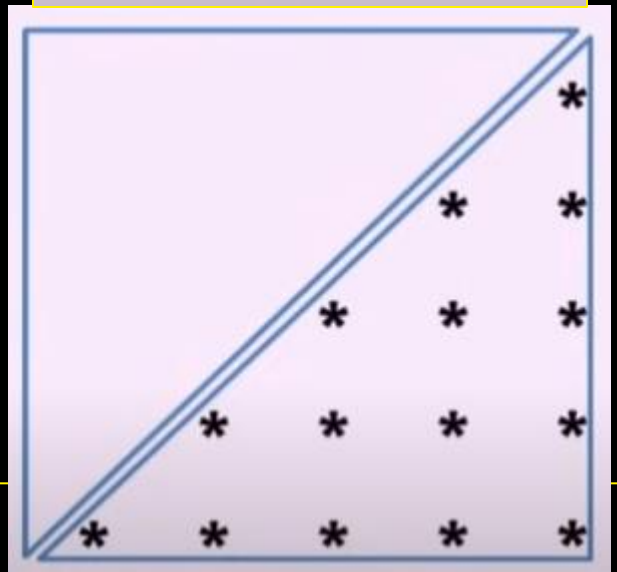
Nested Loop

```
#decreasing triangle pattern

n=5
for i in range(n):
    for j in range(i,n):
        print("*",end=' ')
    print()
```



Nested Loop

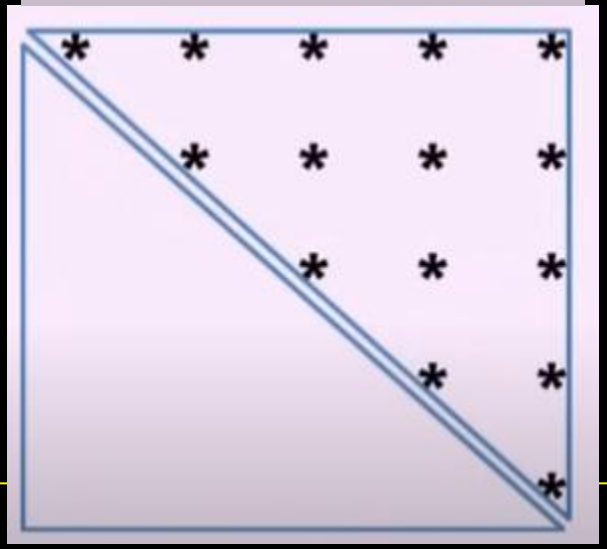
```
#decreasing space and increasing star pattern

n=5
for i in range(n):
    for j in range(i,n):
        print(" ",end=' ')
    for k in range(i+1):
        print("*",end=' ')
    print()
```



Nested Loop

#increasing space and decreasing star pattern

```
n=5
for i in range(n):
    for j in range(i+1):
        print(" ",end=' ')
    for k in range(i,n):
        print("*",end=' ')
    print()
```



Nested Loop

1	1	*
12	22	**
123	333	***
1234	4444	****
12345	55555	*****

Nested Loop

```
rows=int(input("Enter the row value"))
for i in range(1,rows+1):
    for j in range(1,i+1):
        print(j,end="")
    print()
```

```
1
12
123
1234
12345
```

```
rows=int(input("Enter the row value"))
for i in range(1,rows+1):
    for j in range(1,i+1):
        print(i,end="")
    print()
```

```
1
22
333
4444
55555
```

```
rows=int(input("Enter the row value"))
for i in range(1,rows+1):
    for j in range(1,i+1):
        print("*",end="")
    print()
```

```
*
**
***
****
*****
```

Nested Loop

```
n = 5
for i in range(n, 0, -1):
    for j in range(i):
        print("*", end=" ")
    print()
```

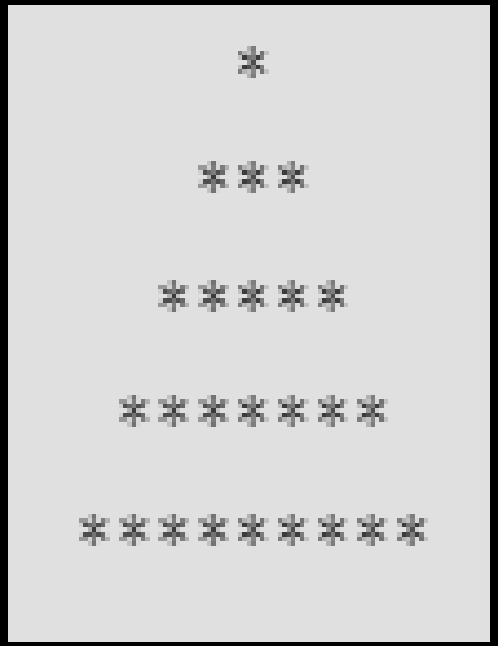
```
* * * * *
* * * *
* * *
* *
*
```

```
n = 6
for i in range(n, 0,-1):
    for j in range(i, 0, -1):
        print(j, end=" ")
    print()
```

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

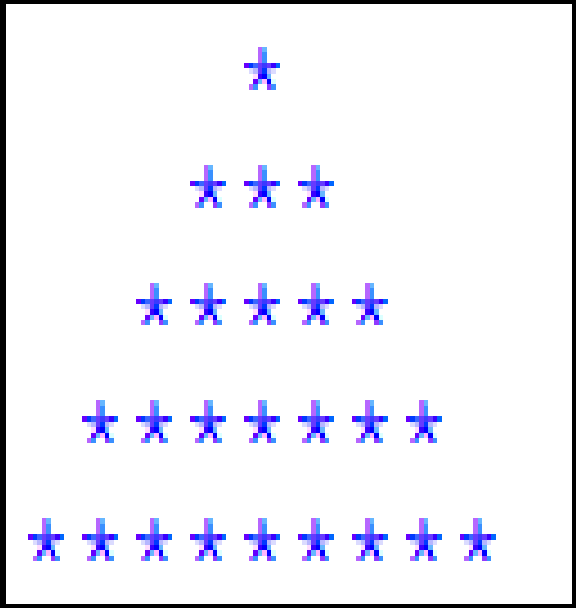
Nested Loop

```
n=5
for i in range(1,n+1):

    for j in range(n-i):
        print(" ",end="")

    for k in range(1,2*i):
        print("*",end="")

    print()
```



Nested Loop

```
n = 5 # Number of rows
for i in range(n):
    print(' ' * (n - i - 1) + '*' * (2 * i + 1))
```



Nested Loop

```
n=int(input("enter the number of rows"))
k=0
for i in range(n):
    k=k+i
#print(k)
m=n+k
#print(m)
for i in range(n):
    for j in range(i+1):
        print(format(m,"<3"),end=" ")
        m=m-1

    print()
```

```
15
14  13
12  11  10
9   8   7   6
5   4   3   2   1
```

format(m, "<3")

Uses Python’s format() function to convert the variable m into a left-aligned string with minimum width 3 characters.

```
# Simple examples with numbers and strings

num = -42
text = "Hi"

# Left-align text: padded to width 6
print(f"Left : [{format(text, '<6')}]")

# Right-align text: padded to width 6
print(f"Right: [{format(text, '>6')}]")

# Center-align text: padded to width 6
print(f"Center:[{format(text, '^6')}]")

# '=' alignment with a number: pads between sign and digits
print(f"Equal :[{format(num, '=6')}]")
```

```
Left : [Hi    ]
Right: [    Hi]
Center:[  Hi  ]
Equal :[-   42]
```

Nested Loop

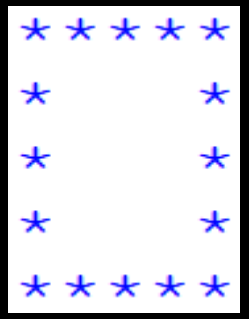
Floyd's triangle is a triangular array of natural numbers used in computer science education. It is named after Robert Floyd. It is defined by filling the rows of the triangle with consecutive numbers, starting with a 1 in the top left corner:

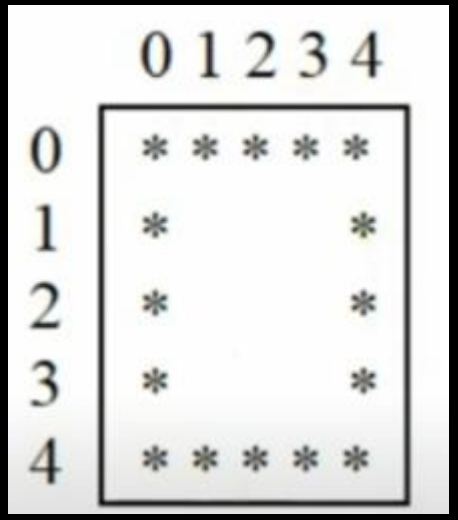
```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

```
n=int(input("enter the number of rows"))
number=1
for i in range(0,n):
    for j in range(0,i+1):
        print(format(number,"<3"),end='')
        number=number+1
    print()
```

Nested Loop

```
n=5
for i in range(n):
    for j in range(n):
        if i==0 or i==n-1 or j==0 or j==n-1:
            print("*",end='')
        else:
            print(" ",end="")
    print()
```





4. JUMP STATEMENTS

4. JUMP STATEMENTS

1. break STATEMENT

2. continue STATEMENT

1. break STATEMENT

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

1. break STATEMENT in FOR loop

```
y=6
for i in range(0,10):
    if i==y:
        print("Thank You")
        break
    else:
        print(i)
print("End of Prg")
```

OUTPUT

```
0
1
2
3
4
5
Thank You
End of Prg
```

break in While Loop

```
number=2
while True:
    guess = int(input("Enter a number"))

    if guess < number:
        print('Your guess is too low.')
    elif guess > number:
        print('Your guess is too high.')
    else:
        print('Great! You have correctly guessed the number')
        break
```

Output

```
Enter a number45
Your guess is too high.
Enter a number3
Your guess is too high.
Enter a number1
Your guess is too low.
Enter a number2
Great! You have correctly guessed the number
```

2. continue STATEMENT

The continue statement in Python 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. The continue statement can be used in both while and for loops.

2. continue STATEMENT

```
for i in range(0,5):
    if i==2:
        continue
    print(i)
```

```
0
1
3
4
```

when i value becomes 2 the print statement gets skipped, continue statement goes for next iteration, hence in the output 2 is not printed

2. continue STATEMENT

```
i=0
while (i<5):
    i=i+1
    if i==2:
        continue
    print(i)
```

```
1
3
4
```

when i value becomes 2 the print statement gets skipped, continue statement goes for next iteration, hence in the output 2 is not printed

pass STATEMENT

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.

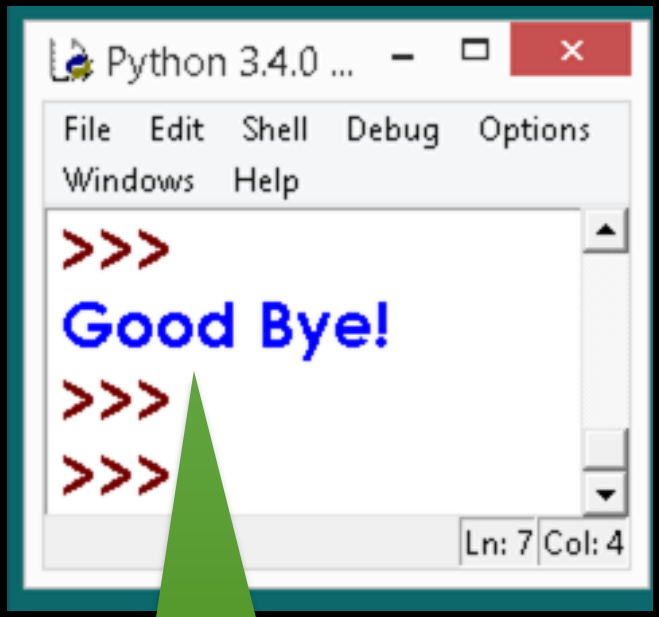
The pass statement is a *null* operation; nothing happens when it executes.

The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

pass STATEMENT

```
for i in range(0,10):
    pass
print("Good Bye!")
```

pass in loop



pass in loop
has no output

pass STATEMENT

```
i=0
while(i<10):
    i=i+1
    if i==3:
        pass

print("Good Bye!")
```

```
Good Bye!
```

pass in loop

pass in loop
has no output

Break, continue, pass

break

With the break statement we can stop the loop even if the while condition is true

continue

With the continue statement we can stop the current iteration, and continue with the next

pass

Does nothing at all: it’s an empty statement placeholder

Difference Between break and continue

BREAK	CONTINUE
It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
It causes early termination of loop.	It causes early execution of the next iteration.
'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.

Robust Test Cases

- **Robust Test Cases** are designed to thoroughly test a function by considering:
- **Normal cases** – typical valid inputs
- **Boundary cases** – values on the edge of valid input range
- **Invalid cases** – values outside the valid range (unexpected or incorrect inputs)

- Calculate **BMI** and check whether the perso is **underweight, normal, or overweight**.
  - Get weight (kg) and height(m) as input
  - bmi = weight / (height * height)
  - Round bmi to 2 decimal places
  - if bmi is less than 18.5, person is underweight
  - If bmi >= 18.5 and bmi <= 24.9, person is normal.
  - Otherwise if bmi >24.9, person is overweight.
  - Perform only one validation: check that inputs are **not negative**.

```
# Get user input
weight = float(input("Enter your weight in kg: "))
height = float(input("Enter your height in meters: "))

# Check for negative values
if weight < 0 or height < 0:
    print("Error: Weight and height must be positive numbers.")
else:
    # Calculate BMI
    bmi = weight / (height * height)
    print("Your BMI is:", round(bmi, 2))

    # Determine BMI category
    if bmi < 18.5:
        print("You are underweight.")
    elif bmi <= 24.9:
        print("Your weight is normal.")
    else:
        print("You are overweight.")
```

Test Cases

Test Case No.	Weight (kg)	Height (m)	BMI	Expected Output	Reason (Boundary Point)
1	50	1.65	18.37	Underweight	Just below 18.5
2	50.3	1.65	18.48	Underweight	Still just below 18.5
3	50.4	1.65	18.51	Normal	More than 18.5
4	67.9	1.65	24.9 **(Boundary Case)**	Normal	Upper limit of normal
5	68	1.65	24.98	Overweight	More than 24.9
6	68.1	1.65	25.01	Overweight	Just above normal (overweight)
7	-45 (**Invalid case)**	1.60		Error: Inputs must be positive numbers.	Weight can't be -ve

Tool to visualize how the code runs

https://pythontutor.com/render.html#mode=display

Thank You