ITA6017
Python Programming

Dr. Arun Pandian J

Module 2: Python Operators, Expressions and Flow controls

All Operations and simple expressions, Conditional blocks using if, else and elif, Simple for loops in python, For loop using ranges, Use of while and do while-loop in python, Loop manipulation using pass, continue, break and else.

Operators

- In computer programming languages operators are special symbols which represent computations, conditional matching etc.
- The value of an operator used is called operands.
    - **Arithmetic operators**
    - **Relational or Comparative operators**
    - **Logical operators**
    - **Assignment operators**
    - **Conditional operator**

Arithmetic operators

- An arithmetic operator is a mathematical operator that takes two operands and performs a calculation on them.

Operator - Operation	Examples	Result
Assume a=100 and b=10. Evaluate the following expressions		
+ (Addition)	>>> a + b	110
- (Subtraction)	>>>a – b	90
* (Multiplication)	>>> a*b	1000
/ (Divisioin)	>>> a / b	10.0
% (Modulus)	>>> a % 30	10
** (Exponent)	>>> a ** 2	10000
// (Floor Division)	>>> a//30 (Integer Division)	3

Relational or Comparative operators

- A Relational operator is also called as Comparative operator which checks the relationship between two operands.

Operator - Operation	Examples	Result
Assume the value of a=100 and b=35. Evaluate the following expressions.		
== (is Equal)	>>> a==b	False
> (Greater than)	>>> a > b	True
< (Less than)	>>> a < b	False
>= (Greater than or Equal to)	>>> a >= b	True
<= (Less than or Equal to)	>>> a <= b	False
!= (Not equal to)	>>> a != b	True

Logical operators

- Logical operators are used to perform logical operations on the given relational expressions.

Operator	Example	Result
Assume a = 97 and b = 35, Evaluate the following Logical expressions		
or	>>> a>b or a==b	True
and	>>> a>b and a==b	False
not	>>> not a>b	False i.e. Not True

Assignment operators

- = is a simple assignment operator to assign values to variable.

Operator	Description	Example
Assume x=10		
=	Assigns right side operands to left variable	>>> x=10 >>> b="Computer"
+=	Added and assign back the result to left operand i.e. x=30	>>> x+=20 # x=x+20
-=	Subtracted and assign back the result to left operand i.e. x=25	>>> x-=5 # x=x-5
*=	Multiplied and assign back the result to left operand i.e. x=125	>>> x*=5 # x=x*5
/=	Divided and assign back the result to left operand i.e. x=62.5	>>> x/=2 # x=x/2

Conditional Operator

- Ternary operator is also known as conditional operator that evaluate something based on a condition being true or false.
- It simply allows testing a condition in a single line

Variable Name = [on_true] if [Test expression] else [on_false]

```
a=10
b=20
min = a if a < b else b
```

Task:

Reads two integers from user, a and b. Add code to print three lines where:

- The first line contains the sum of the two numbers.
- The second line contains the difference of the two numbers (first - second).
- The third line contains the product of the two numbers.

Task:

Reads two integers, a and b, from input. Add logic to print two lines. The first line should contain the result of integer division, a//b. The second line should contain the result of float division, a/b.

No rounding or formatting is necessary.

Simple Expression

- An expression is a combination of operators and operands that is interpreted to produce some other value.
- In any programming language, an expression is evaluated as per the precedence of its operators.
- So that if there is more than one operator in an expression, their precedence decides which operation will be performed first.

Example:

x = 15 + 1.3

add = x + y

sub = x - y

pro = x * y

div = x / y

c = a + int(b)

p = (a + b) >= (c - d)

Variables and Expressions

Constants

- Fixed values such as numbers, letters, and strings are called "constants" because their value does not change

  Numeric constants are as you expect

  String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Variables

A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

Programmers get to choose the names of the variables

You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
```

x	12.2
y	14

Variables

A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

Programmers get to choose the names of the variables

You can change the contents of a variable in a later statement

x = 12.2
y = 14
x = 100

x	~~12.2~~ 100
y	14

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters and numbers and underscores
- Case Sensitive

Good: spam eggs spam23 _speed
Bad: 23spam #sign var.12
Different: spam Spam SPAM

Reserved Words

You cannot use reserved words as variable names / identifiers

False	**class**	**return**	**is**		**finally**
None	**if**		**for**	**lambda**	**continue**
True	**def**	**from**	**while**	**nonlocal**	
and	**del**	**global**	**not**	**with**	
as	**elif**	**try**		**or**	
	yield				
assert	**else**	**import**	**pass**		
break	**except**	**in**		**raise**	

Sentences or Lines

x = 2	Assignment statement
x = x + 2	Assignment with expression
print(x)	Print statement

Variable Operator Constant Reserved Word

Assignment Statements

- We assign a value to a variable using the assignment statement (=)

- An assignment statement consists of an expression on the right-hand side and a variable to store the result

x = 3.9 * x * (1 - x)

A variable is a memory location used to store a value (0.6)

x 0.6

0.6 0.6

x = 3.9 * x * (1 - x)

0.4

0.936

The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) x.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).

x ~~0.6~~ 0.93

0.6 0.6

x = 3.9 * x * (1 - x)

0.4

0.936

The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

Numeric Expressions

Because of the lack of mathematical symbols on computer keyboards - we use "computer-speak" to express the classic math operations

Asterisk is multiplication

Exponentiation (raise to a power) looks different from in math.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Order of Evaluation

When we string operators together - Python must know which one to do first

This is called “operator precedence”

Which operator “takes precedence” over the others?

x = 1 + 2 * 3 - 4 / 5 ** 6

Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

Parenthesis are always respected

Exponentiation (raise to a power)

Multiplication, Division, and Remainder

Addition and Subtraction

Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right

1 + 2 ** 3 / 4 * 5

↓

1 + 8 / 4 * 5

↓

1 + 2 * 5

↓

1 + 10

↓

11

Operator Precedence

Parenthesis
Power
Multiplication
Addition
Left to Right

Remember the rules top to bottom

When writing code - use parenthesis

When writing code - keep mathematical expressions simple enough that they are easy to understand

Break long series of mathematical operations up to make them more clear

Task: x = 1 + 2 * 3 - 4 / 5

Task:

Read the inputs as dictionary containing key/value pairs of name:[marks] for a list of students. Print the average of the marks array for the student name provided, showing 2 places after the decimal.

Example

marks key:value pairs are

'alpha': [20, 30, 40]

'beta': [30, 50, 70]

query_name = 'beta'

The **query_name** is 'beta'. beta's average score is $(30 + 50 + 70)/3 = 50.0$.

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. **BRANCHING OR JUMPING STATEMENTS**

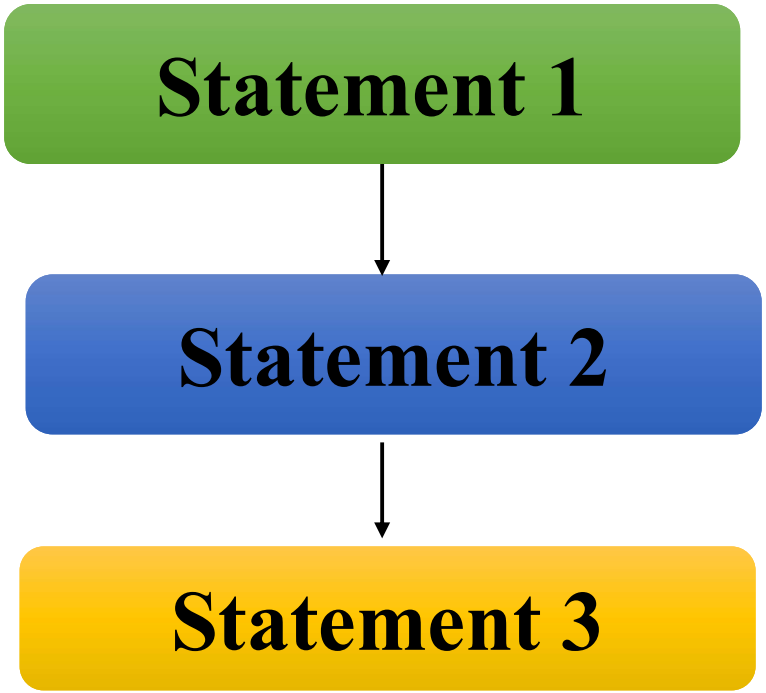
1. SEQUENCE

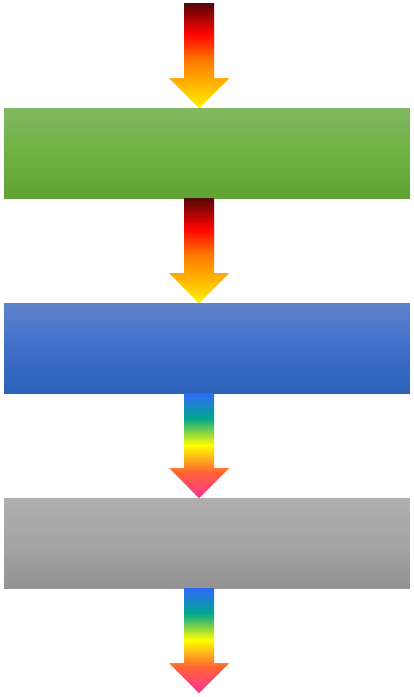
Sequence is the default control structure; instructions are executed one after another.

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

1. SEQUENCE – FLOW CHART

1. SEQUENCE – FLOW CHART





SEQUENCE

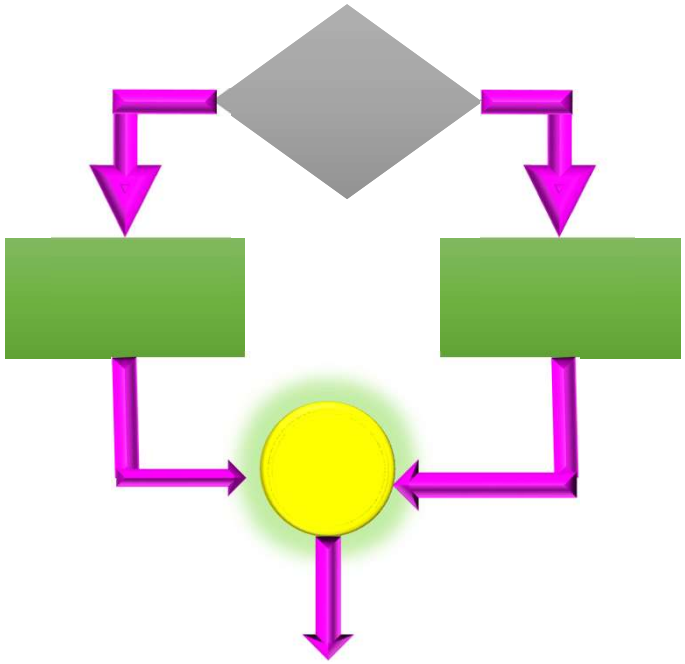
1. SEQUENCE - PROGRAM

1. SEQUENCE - PROGRAM

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

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

2. SELECTION



SELECTION

2. SELECTION

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

: Colon Must

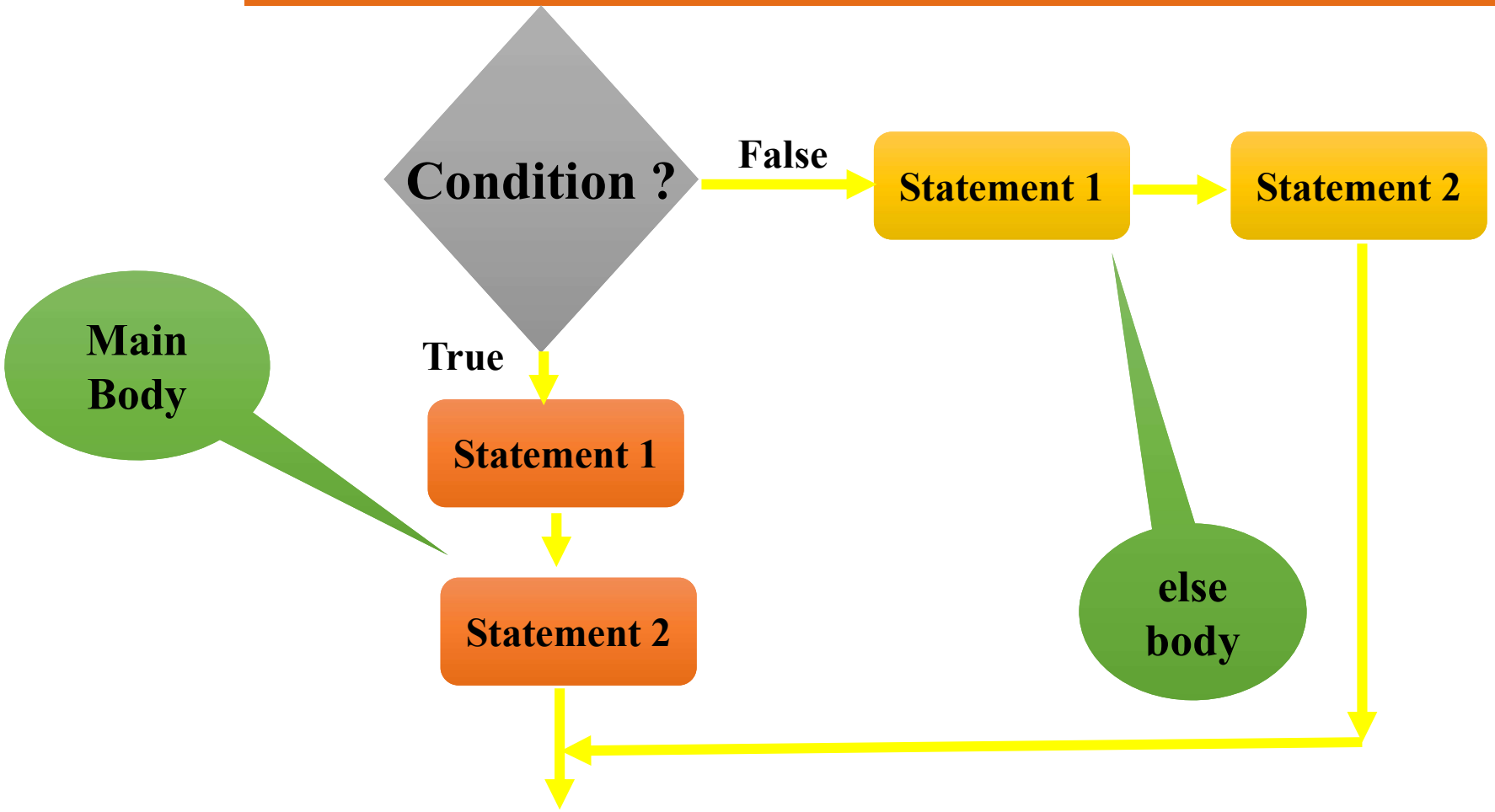
```
if first condition:
    first body
elif second condition:
    second body
elif third condition:
    third body
else:
    fourth body
```

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

Python 3.4.0: ifelse.py - C:\Python34\ifelse.py

File Edit Format Run Options Windows Help

```
def if_example():
    a = 5
    if (a <10):
        print ("5 is less than 10")
    print ("Statement after if statement")
if_example()
```

Ln: 8 Col: 0

else is missing, it is an optional statement

OUT PUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
5 is less than 10
Statement after if statement
>>>
```

Ln: 14 Col: 4

CONDITIONAL CONSTRUCT

EXAMPLE – if else STATEMENT

EXAMPLE – if else STATEMENT

Python 3.4.0: ifelse.py - C:\Python34\ifelse.py

File Edit Format Run Options Windows Help

```
def if_else_example():
    age = 15
    if (age >=18):
        print ("Elegible for Voting")
    else:
        print("Not Eligible for Voting")
    print ("Statement after if statement")
if_else_example()
```

Ln: 13 Col: 0

: Colon Must

else is used

OUT PUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
>>>
Not Eligible for Voting
Statement after if statement
>>>
```

Ln: 18 Col: 4

CONDITIONAL CONSTRUCT

EXAMPLES – if elif STATEMENT

EXAMPLES – if elif STATEMENT

Python 3.4.0: ifelse.py - C:\Python34\ifelse.py

File Edit Format Run Options Windows Help

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

Ln: 12 Col: 0

READ AS

18 is less than age and 18 is less than 60

OUTPUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
No Discount
>>>
```

Ln: 9 Col: 4

PROGRAM LIST ON if CONTSTUCT

PROGRAM LIST ON if CONTSTUCT

BELOW AVERAGE PROGRAMS

1. Write a PYTHON program that reads a value of n and check the number is zero or non zero value.
2. Write a PYTHON program to find a 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.

PROGRAM LIST ON if CONTSTUCT

AVERAGE PROGRAMS

5. Write a PYTHON program to evaluate the student performance

If % is >=90 then Excellent performance

If % is >=80 then Very Good performance

If % is >=70 then Good performance

If % is >=60 then average performance

else Poor performance.

6. Write a PYTHON program to find largest of three numbers.

7. Write a PYTHON program to find smallest of three numbers

PROGRAM LIST ON if CONTSTUCT

ABOVE AVERAGE PROGRAMS

8.Write a PYTHON program to check weather number is even or odd.

9.Write a PYTHON program to check a year for leap year.

10. A company insures its drivers in the following cases:

- If the driver is married.
- If the driver is unmarried, male and above 30 years of age.
- If the driver is unmarried, female and above 25 years of age.

PROGRAM LIST ON if CONTSTUCT

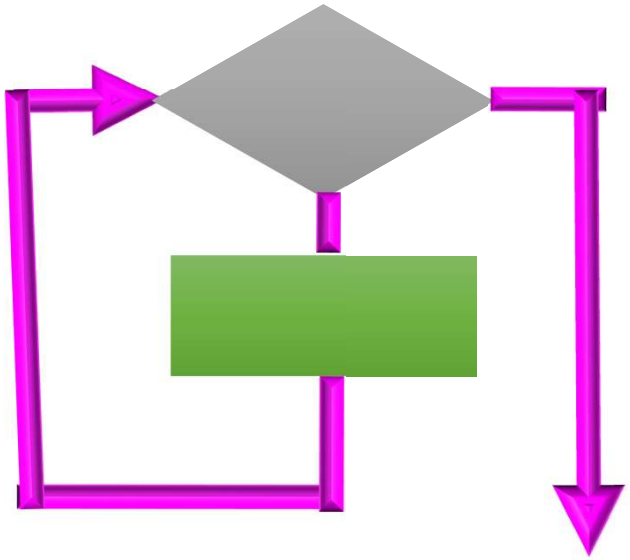
ABOVE AVERAGE PROGRAMS

In all the other cases, the driver is not insured. If the marital status, sex and age of the driver are the inputs,

Write a PYTHON program to determine whether the driver is insured or not

***

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,

for loop

while loop

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

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

*Python 3.4.0: for_natno.py - C:/Python34...

File Edit Format Run Options Windows Help

```
def for_loop_example():
    numbers=[765,23,56,89,14,78]
    for i in numbers:
        print(i)
for_loop_example()
```

Ln: 8 Col: 0

Till the list exhaust for loop will continue to execute.

OUTPUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
>>>
765
23
56
89
14
78
>>>
```

Ln: 8 Col: 0

for LOOP

range KEYWORD

for LOOP - range KEYWORD

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

range*(start, stop, step)*

```
for n in range(3,6):
  print(n)
```

O R

```
x = range(3, 6)
for n in x:
  print(n)
```

for LOOP - range KEYWORD

#Generating series of numbers

Python 3.4.0: fornat.py - C:/Python34/forn...

File Edit Format Run Options Windows Help

```
def for_loop_example():
    n=int(input("Enter number: "))
    for i in range(0,n):
        print(i)
for_loop_example()
```

Ln: 2 Col: 27

OUTPUT

Python 3.4.0 S...

File Edit Shell Debug Options Windows Help

```
Enter number: 4
0
1
2
3
>>>
```

Ln: 16 Col: 4

for LOOP - range KEYWORD

#Generating even numbers

*Python 3.4.0: fornat.py - C:/Python34/for...

File Edit Format Run Options Windows Help

```
def for_loop_example():
    n=int(input("Enter number: "))
    for i in range(0,n,2):
        print(i)
for_loop_example()
```

Ln: 6 Col: 0

OUTPUT

Python 3.4.0 ...

File Edit Shell Debug Options Windows Help

```
Enter number: 6
0
2
4
>>>
```

Ln: 12 Col: 4

for LOOP – len() FUNCTION

for LOOP - range KEYWORD

print string character by character

*Python 3.4.0: forstr.py - C:/Python34/fo...

File Edit Format Run Options Windows Help

```
#printing string char by char
def for_loop_example():
    name=input("Enter string: ")
    for i in range(0,len(name)):
        print(name[i])
for_loop_example()
```

Ln: 10 Co

OUTPUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
Enter string: Sainik
S
a
i
n
i
k
>>>
```

Ln: 21 Col: 4

else statement in loop

else statement in loop

else can be used in for and while loops the else body will be executed as and when the loop's conditional expression evaluates to false

Python 3.4.0: breakinloop.py - C:/Python34/breakinloop.py

File Edit Format Run Options Windows Help

```
def break_in_loop():
    for i in range(0,5):
        print (i)
    else:
        print("Finally! Else of for executed")
break_in_loop()
```

Ln: 7 Col: 0

OUTPUT

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
========
>>>
0
1
2
3
4
Finally! Else of for executed
>>>
```

Ln: 11 Col: 4

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

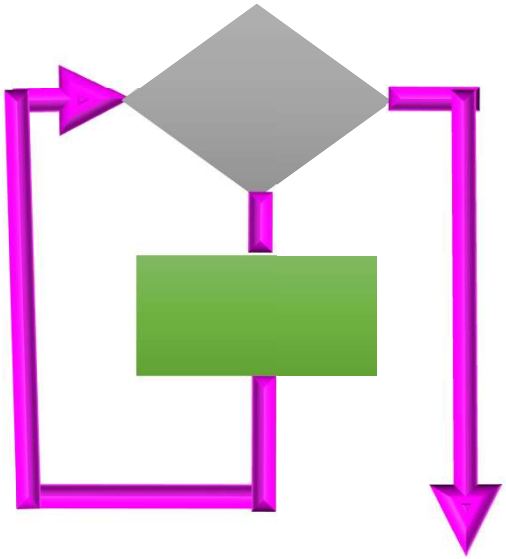
Where, loop body contain the single statement or set of statements (compound statement) or an empty statement.

Contd..

while loop

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

FLOW CHART



while loop

while loop – Programming example

while loop - programs

Natural Numbers generation

*Python 3.4.0: nat_while.py - C:/Python34/nat_wh...

File Edit Format Run Options Windows Help

```
#Program to generate natural nos
def gen_nat_no_while():
    i=1
    n=int(input("Enter the limit : "))
    while(i<=n):
        print(i)
        i+=1
gen_nat_no_while()
```

Ln: 11 Col: 0

OUTPUT

Python 3.4.0 S...

File Edit Shell Debug Options Windows Help

```
Enter the limit : 5
1
2
3
4
5
```

Ln: 26 Col: 4

while loop - programs

Calculating Sum of Natural Numbers

*Python 3.4.0: while_llop.py - C:/Python34/w...

File Edit Format Run Options Windows Help

```
#sum of Natural numbers
def while_loop_example():
    sum1 = 0
    count = 1
    while (count < 10):
        sum1 = sum1 + count
        count = count + 1
    print (count) # should be 10
    print (sum1) # should be 45
while_loop_example()
```

Ln: 12 Col: 0

OUTPUT

Python 3.4.0 ...

File Edit Shell Debug Options Windows Help

```
10
45
```

Ln: 17 Col: 4

while loop - programs

#Generating Fibonacci numbers

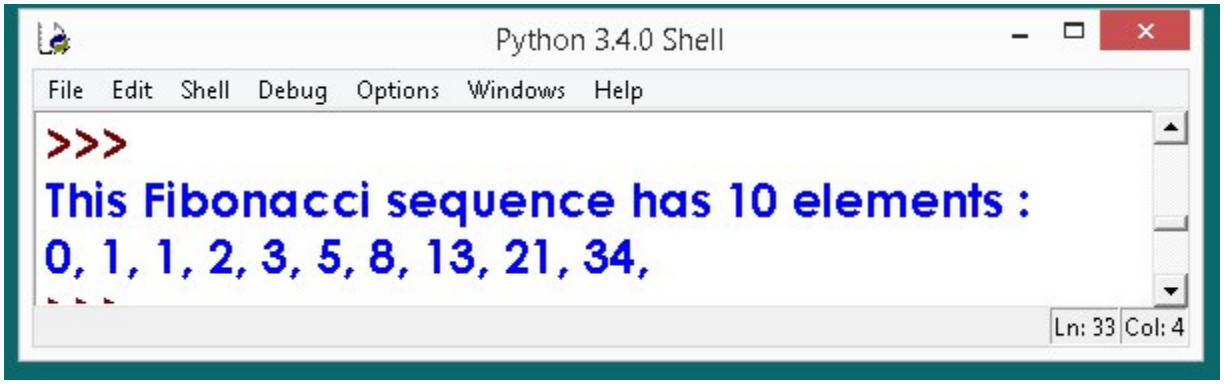
File Edit Format Run Options Windows Help

```
def fibo_numbers():
    length = 10
# The first two values
    x = 0
    y = 1
    iteration = 0
# Condition to check if the length has a valid input
    if length <= 0:
        print("Please provide a number greater than zero")
    elif length == 1:
        print("This Fibonacci sequence has {} element".format(length), ":")
        print(x)
    else:
        print("This Fibonacci sequence has {} elements".format(length), ":")
    while (iteration < length):
        print(x, end=', ')
        z = x + y
        # Modify values
        x = y
        y = z
        iteration += 1
fibo_numbers()
```

while loop - programs

#Generating Fibonacci numbers

OUTPUT



4. BRANCHING OR JUMPING STATEMENTS

4. BRANCHING OR JUMPING STATEMENTS

Python has an unconditional branching statements and they are,

1. break STATEMENT

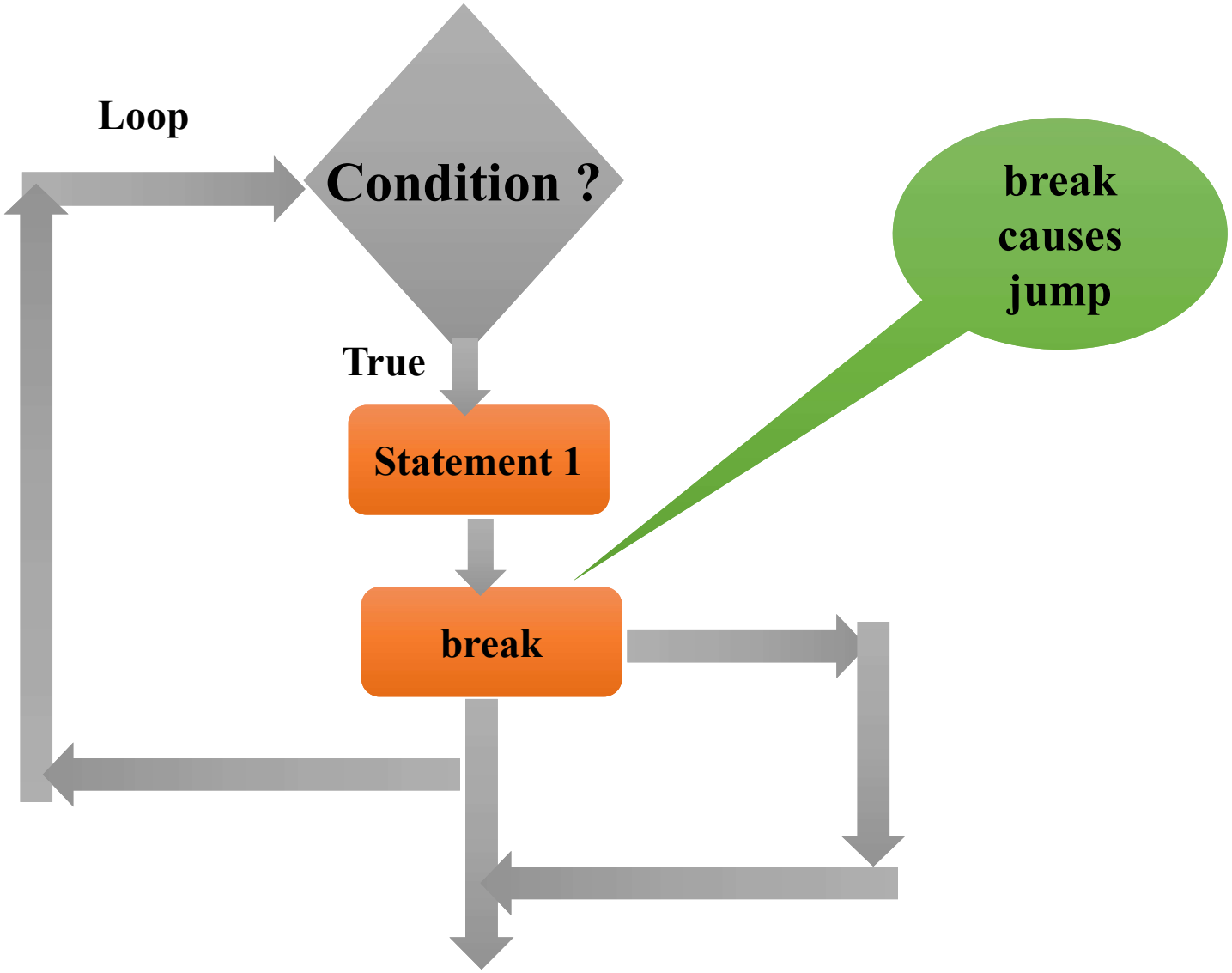
2. continue STATEMENT

4. BRANCHING OR JUMPING STATEMENTS

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



1. break STATEMENT

Python 3.4.0: breakexample.py - C:\Python34\bre...

File Edit Format Run Options Windows Help

```
def break_example():
    y=5
    for i in range(0,y+1):
        if i == y:
            print("Thank you!")
            break
        else:
            print(i)
    print("End of Prg")
break_example()
```

Ln: 11 Col: 0

OUT PUT

Python ...

File Edit Shell Debug Options Windows Help

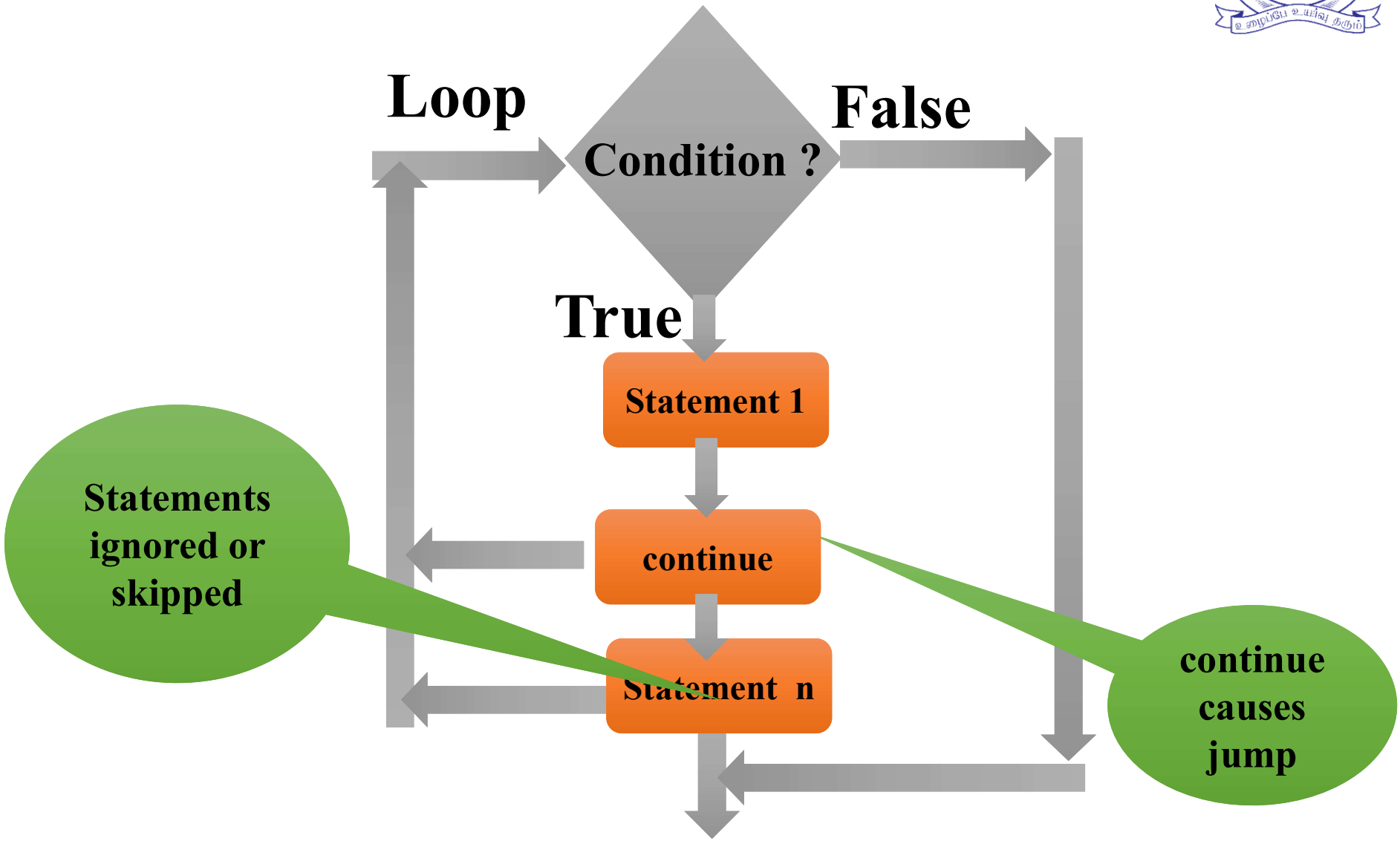
```
>>>
0
1
2
3
4
Thank you!
End of Prg
>>>
```

Ln: 12 Col: 4

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



2. continue STATEMENT

Python 3.4.0: continueexa.py - C:/Pyth...

File Edit Format Run Options Windows Help

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

continue_in_loop()
```

Ln: 8 Col: 0

File Edit Shell Debug Options Windows Help

```
>>>
0
1
3
4
>>>
```

Ln: 9 Col: 4

when i value becomes 2 the print statement gets skipped, continue statement goes for next iteration, hence in the out put 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

Python 3.4.0: passexampl...

File Edit Format Run Options Windows Help

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

Ln: 6 Col: 0

pass in loop

Python 3.4.0 ...

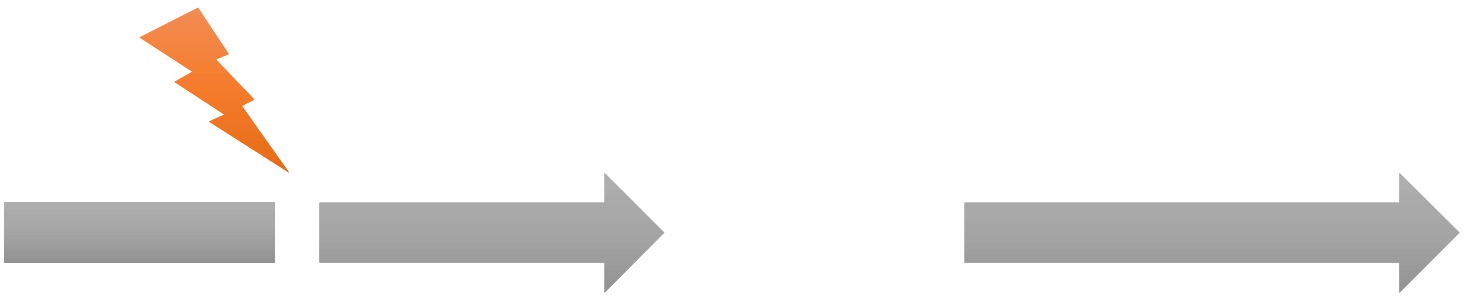
File Edit Shell Debug Options Windows Help

```
>>>
Good Bye!
>>>
>>>
```

Ln: 7 Col: 4

pass in loop has no output

Difference Between break and continue



break **continue**

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.

Thank You