Unit -2

Topics to be Covered

Types, Operators and Expressions: Types - Integers, Strings, Booleans; Operators- Arithmetic Operators, Comparison (Relational) Operators, Assignment Operators, Logical Operators, Bitwise Operators, Membership Operators, Identity Operators, Expressions and order of evaluations Control Flow- if, if-elif-else, for, while, break, continue, pass

Python Data Types

Python Data Types are classified as “***Core Data Types***” , “***Compound Data Types***” and “***Boolean Data Type***”. Under the Core data Types we have Numbers and Strings. Under the Compound Data Types, we have Lists, Tuples, Dictionary, and set etc. The Numbers and Strings represent the Numeric and Textual values, respectively. Under the Boolean Data, a variable can contain any one of the two values: True or False.

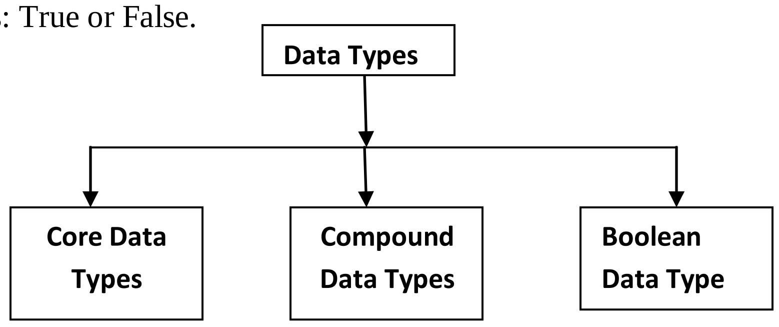


Fig 1: Data Types in Python

Core Data Types

Numbers: Integers, floating point numbers and complex numbers are fall under the Number category. These are defined as “int”, “float”, and “complex” class in Python. We can use the type function to know type or class of a variable. Integers can be of any length, it is only limited by the memory variable. A floating point number is accurate up to 15 decimal points.

For example, 1 is an Integer. 1.0 is floating point number. The complex numbers are written in form, x+yj, where x is the Real part and y is the imaginary part.

Procedure to know the Type of the Data:

```
>>> x=12

>>> print("class of x is:",type(x))

('class of x is:', <type 'int'>)

>>> y=13.4
>>> print("class of y is:",type(y))

('class of y is:', <type 'float'>)

>>> a=12+13j

>>> print("class of a is:",type(a))

('class of a is:', <type 'complex'>)

>>> z=True
```

Strings: String is a sequence of characters. We can use single quotes or double quotes to represent the strings. Multiline strings are denoted with Triple Quotes, ' ' ' or " " ".

Example Program

```
s="Hello Python Programmer"
s1='''It is easy to write and read.It is easy to write and read.It is easy to write and read.It is easy to write and read.'''
print(s)
print(s1)
```

Output:

```
Hello Python Programmer
It is easy to write and read.It is easy to write and read.It is easy to write and read.It is easy to write and
read.
```

.
We can get the character at the specified position by its index. We can also get the substring from the index by specifying the range of the substring. The index always starts from zero. See the following example:

```
s="Hello Python Programmer"
print(s)
print(s[2]) # obtaining the character at position 2
print(s[3:7]) # retrieving substring using the from index 3 to 7
```

Output:

```
Hello Python Programmer
l
lo P
```

Boolean Data Type

A Boolean variable can reference one of two values: **True or False**. Boolean variables are commonly used as flags, which indicate whether specific conditions exist.

Example Program: **booltest.py**

```
b=False
print "The value of b is:",b
print "class of b is:",type(b)
b=True
print "The value of b is:",b
print "class of b is:",type(b)
```

Output:

```
The value of b is: False
class of b is: <type 'bool'>
The value of b is: True
class of b is: <type 'bool'>
```

Compound Data Types

List, Tuple, Sets and Dictionaries are fall under the Compound Data Types. We will discuss about these in the 3rd unit.

List: Is the Order collection of different types of items. All the items need not be of same type. These are mutable, that means the list can be modified using the index of the item or using the predefined methods such as "***append()***", "***sort()***", "***pop()***", and "***reverse()***". The list is created with square brackets [].

Example:

```
M=[“apple”, 2,45.7,”Sweet”]
M[1]=”Orange”    # it is a valid operation, we can change an item to another item
```

After change

M=[“apple”, “**Orange”**,45.7,”Sweet”]

Tuples: It is roughly like a list, but is immutable, that means, the tuple cannot be modified once it is created. We can get the index of an item. We can also get the frequency of the item using the function like "***index()*** "and "***count()***".. This is created with parentheses ().
Example:

```
T=(1,2,3,4)
T[1]=”Apple” # not possible to change, it is an error
```

Type Conversion

Each Python type comes with a built-in function that attempts to convert values of another type into that type. The **int(ARGUMENT)** function, for example, takes any value and converts it to an integer, if possible, or complains otherwise:

Example:
The following statement converts float value into integer.
int(12.3) ->12
The following statement converts integer value float integer.
float(12) -> 12.0

It may seem odd that Python distinguishes the integer value 1 from the floating-point value 1.0. They may represent the same number, but they belong to different types. The reason is that they are represented differently inside the computer.

Operators

Expression: An Expression is a combination of operators and operands that computes a value when executed by the Python interpreter. In python, an expression is formed using the usual mathematical operators and operands (sometimes can be values also).

For example, to add two numbers that are stored in operands x and y is written as, x+y. Where + is the Arithmetic addition operator. The operator in Python are classified as follow:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operator
5. Assignment Operators
6. Membership Operators
7. Identity Operators

1. Arithmetic Operators

These operators are used to perform operations such as addition, subtraction, multiplication, division and modulo division. For example, x=7, and y=3.

Operator	Meaning	Example
+	Addition-Used to perform arithmetic addition	x+y, results in 10
-	Subtraction-Used to perform arithmetic subtraction	x-y, results in 4
*	Multiplication-Used to perform multiplication	x*y, results in 21
/	Division-Used to perform division	x/y , results in 2
%	Modulus-Used to perform modulus operation (remainder)	x%y, results in 1
//	Used to perform floor division (floor value)	x//y, results in 2
**	Exponent- Used to raise operand on left to the power of operand on right	x**y, 343

Example program:

Op.py

Source Code	**Output**
x=input("Enter value of x :")	Enter value of x :7
y=input("Enter value of y :")	Enter value of y :3
print " --------------------- "	-----------------------
print " Addition is:",(x+y)	Addition is: 10
print " Subtraction is:",(x-y)	Subtraction is: 4
print " Multiplication is:",(x*y)	Multiplication is: 21
print " Division is:",(x/y)	Division is: 2
print " Modulus is:",(x%y)	Modulus is: 1
print " Floor Division is:",(x//y)	Floor Division is: 2
print " Exponent is:",(x**y)	Exponent is: 343
print " --------------------- "	-----------------------

2. Comparison (Relational) Operators

These operators are used to compare value. The operators can either return True or False according to the condition. The table with following values for x=7 and y=3

Operator	**Meaning**	**Example**
>	Greater Than-Returns True if the left operand is greater than the right, otherwise returns False	x>y, results in True
<	Less Than-Returns True if the left operand is less than the right, otherwise returns False	X<y, results in False
= =	Equal to-Returns True if both are equal, otherwise False	x==y, returns False
!=	Not Equal to- Returns True if both are not equal, otherwise False	x!=y, return True
>=	Greater than or Equal- Returns True if the left operand is greater than or equal to the right, otherwise returns False	x>y, returns True
<=	Less than or Equal- Returns True if the left operand is Less than or equal to the right, otherwise returns False	X<y, returns False

Example Program: **relp.py**

Source Code	**Output**
x=input("Enter value of x ")	Enter value of x 7
y=input("Enter value of y ")	Enter value of y 3
print " --------------------- "	-----------------------
print " Greater than is:",(x>y)	Greater than is: True
print " Less Than is:",(x<y)	Less Than is: False
print " Equality is :",(x==y)	Equality is : False
print " Not equal is:",(x!=y)	Not equal is: True
print " Greater than or equal is:",(x>=y)	Greater than or equal is: True
print " Less than or equal is:",(x<=y)	Less than or equal is: False
print " --------------------- "	-----------------------

3. Bitwise Operators

Bitwise operators act on operands as if they were string of binary digits. The operators operate bit by bit. For example, x=2 (binary value is 10) and y=7 (Binary value is 111). The binary equivalent of the decimal values of x and y will be 10 and 111 respectively.

Operator	Meaning	Example

&	Bitwise AND	X&y=0
		x=010
		y=111
		x&y= 010 (2)

|	Bitwise OR	x|y=7
		x=010
		y=111
		x|y=111 (7)

~	Bitwise Not	~x is ,-3

^	Exclusive OR (XOR)	X^y=5
		x=010
		y=111
		x|y=101 (5)

>>	Shift Right (operand >>no. of bit positions)	x>>1, results 1
<<	Shift Left (operand <<no. of bit positions)	X<<2, 1000 (8)

Example Program: **bitop.py**

Source Code	Out put
x=input("Enter value of x :") y=input("Enter value of y :") print "-----Bitwise Operations------- " print " AND (&) is:",(x&y) print " OR (|) is:",(x|y) print " XOR (^) is:",(x^y) print " Not (~) is:",(~x) print " Shift Right(>>) is:",(x>>1) print " Shift Left (<<)is:",(x<<2) print " --------------------- "	Enter value of x :2 Enter value of y :7 -----Bitwise Operations-------- AND (&) is: 2 OR (|) is: 7 XOR (^) is: 5 Not (~) is: -3 Shift Right(>>) is: 1 Shift Left (<<)is: 8 -----------------------

4. Logical Operator:

There are three logical operators: **and**, **or**, and **not**. The semantics (meaning) of these operators is similar to their meaning in English. For example, x > 0 and x < 10 is true only if x is greater than 0 and less than 10. n % 2 == 0 or n % 3 == 0 is true if either (or both) of the conditions is true, that is, if the number is divisible by 2 or 3. Finally, the not operator negates a boolean expression, so not(x > y) is true if (x > y) is false, that is, if x is less than or equal to y.

Operator	Meaning	Example
and	True if both the operands are True	x **and** y
or	True, if either of the operands is True	x **or** y
not	True if operand false	**not** x

Example Program: **logop.py**

Source Code	Out Put
x=True y=False print " x and y is :",x and y print " x or y is :",x or y print " not x is:",not x	x and y is : False x or y is : True not x is: False

5. Assignment Operator

Assignment operator is used to assign values to the variable. For example, x=5 is simple assignment operator, that assigns value 5 to the to the variable x. There are various compound operators in python like a+=5, which adds value 5 to the variable and later assigns it to variable a. This expressions is equivalent to a=a+5. The same assignment operator can be applied to all expressions that contain arithmetic operators such as, *=, /=, -=, **=,%= etc.

```
x=4
x+=5
>>> x
print “The value of x is:”, x
>>> The value of x 9
```

6. Membership Operators

These operators are used to test whether a value or operand is there in the sequence such as list, string, set, or dictionary. There are two membership operators in python: ***in*** and ***not in***. In the dictionary we can use to find the presence of the key, not the value.

Operator	Meaning	Example
In	True if value or operand is present in the sequence	5 in x
not in	True if value or operand is not present in the sequence	5 not in x

Example Program:

Source Code	Output
#membership operator: in and not in x="Hello Python" # **string**	('H in x', True) ('hello in x', False)

y={1:'a',2:'n',3:'t'} **#dictionary** print ("H in x",'H' in x) print ("hello in x","hello" in x) print ("1 in y",1 in y) **# 1 key key in dictionary** print ("a in y",'a' in y) **#a is value in dictionary**	('1 in y', True) ('a in y', False)

7. **Identity Operators**

These are used to check if two values (variable) are located on the same part of the memory. If the x is a variable contain some value, it is assigned to variable y. Now both variables are pointing (***referring***) to the same location on the memory as shown in the example program.

Operator	Meaning	Example

Is	True if the operands are identical (refer to the same memory)	X=5
		Y=X
		X is Y , returns True

is not	True if the operands are not identical (refer to the same memory)	X=5 #int
		Y=5.0 # float
		X is not Y, returns True

Example Program:

Source Code	**Output**

#indetity operator program
x=[1,2,3] #list
y=[1,2,3] #list
z=x
print ("x is y", x is y) # x and y are different objects
print ("x is z", x is z) # x and z are refering to same object
	('x is y', False)
	('x is z', True)

Expressions and Order of Evaluations

When more than one operator appears in an ***expression***, the order of evaluation depends on the **rules of precedence**. Python follows the same precedence rules for its mathematical operators that mathematics does. The acronym **PEMDAS** is a useful way to remember the order of operations:

- ✓ **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, **2*(3-1)** is 4, and **(1+1)**(5-2)** is 8. You can also use parentheses to make an expression easier to read, as in (minute*100)/60, even though it doesn't change the result.
- ✓ **E**xponentiation has the next highest precedence, so **2 **1+1** is 3 and not 4, and **3*1** 3** is 3 and not 27.

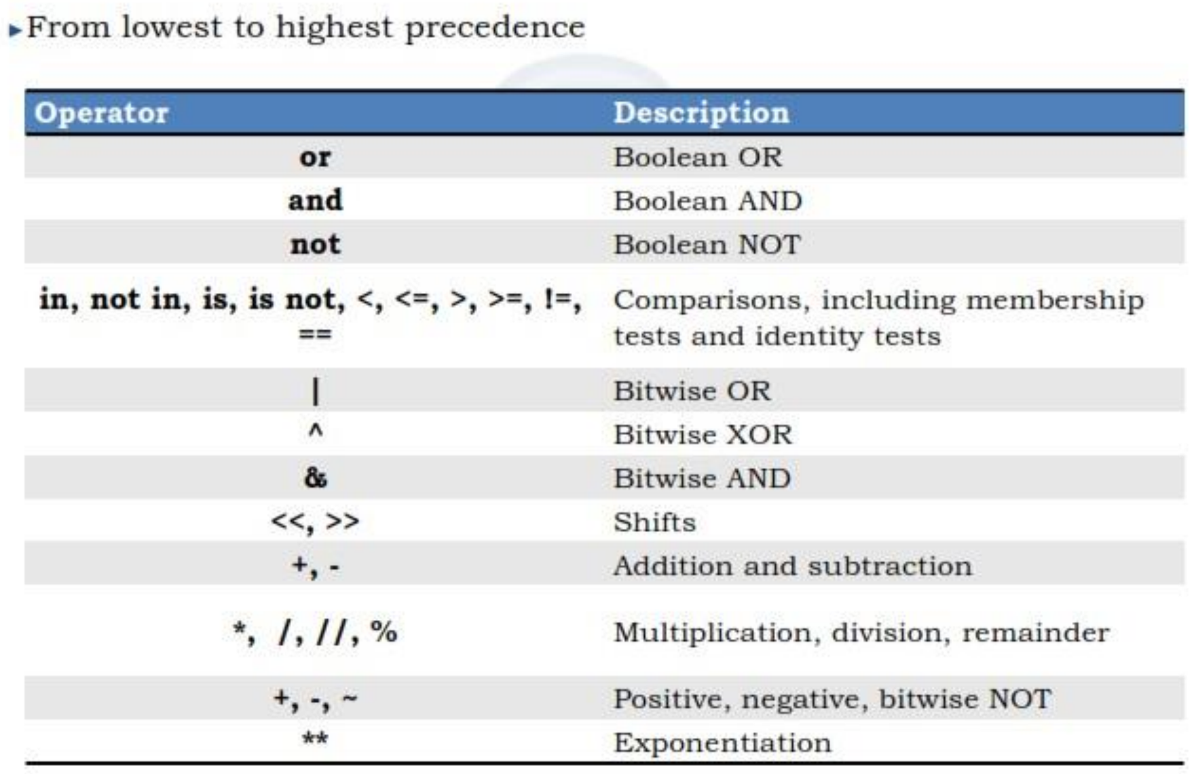
- ✓ **M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So 2 *3-1 yields 5 rather than 4, and 2/3-1 is -1, not 1 (re- member that in integer division, 2/3=0).
- ✓ Operators with the same ***precedence are evaluated from left to right***. If the minute=59, then in the expression **minute*100/60**, the multiplication happens first, yielding 5900/60, which in turn yields 98. If the operations had been evaluated from ***right to left***, the result would have been 59*1 which is wrong. If in doubt, use parentheses.

Example Program:

Source Code	Output

#order of evaluations
x=2
y=3
z=6
print "The value of expression x**y+z is", x**y+z
print "The value of expression (x*y)**2+z-x is", (x*y)**2+z-x
	The value of expression x**y+z is 14
	The value of expression (x*y)**2+z-x is 40
	
	Note: expression in ( ) is executed first, then exponent is calculated, later addition and subtraction at the end

<u>Precedence of the Operators (</u><u>Precedence increase from Top to Bottom</u>**<u>)</u>**



▸From lowest to highest precedence

Operator	Description
or	Boolean OR
and	Boolean AND
not	Boolean NOT
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
***, /, //, %**	Multiplication, division, remainder
+, -, ~	Positive, negative, bitwise NOT
******	Exponentiation

<u>Control Flow:</u> In all most all programming languages the control flow statements are classified as Selection Statements, Loop Statements, or Iterative Statement, and Jump Statements. Under the Selection statements in Python we have ***if, elif*** and ***else*** statement. Under the loop statements we have ***for*** and ***while*** statements. Under the Jump statements we have **break**, **continue** and **pass** statements.

If statement - The **if** statement is used for conditional execution. An **if statement** is followed by a Boolean expression, which is evaluated to either True or False. If Boolean expression is evaluated to True, the ***block*** which contains one or more statements will be executed. Otherwise, the ***block*** followed by the "**else**" statement is executed. The general form of if statement will be as follow in Python:

```
if boolena_expression:
   statement(s)  # block of statements inside if
else:
   statement(s)   # block of statements inside else
```

Example program: Write a Program whether a given Number if even or Odd.

Evenodd.py	Output

#read the number from keyboard
n=input("enter any number :")
if(n%2==0): ***#test the number***
   print ("It is Even")
else:
   print ("It is Odd")
	enter any number :13
	It is Odd
	>>>
	===========================
	enter any number :12
	It is Even
	>>>

if –elif-else statements

This combination of statements is used, whenever; one among multiple alternatives needs to be selected. It selects exactly one block of statements if and only if, one of the Boolean expressions is evaluated to True, otherwise block inside the "**else**" statement will be executed, if present.

General form of if-elif-else will be as follow:

```
if (boolean_expression):
        Block of statements
elif(boolean_expression):
        Block of statements
elif(boolean_expression):
else:
```

Block of statement

Write a Python program to check the whether a given character is Vowel or Consonant.

Vowel.py	output

#***vowel or Consonant***
ch='i'
if ch=='a' or ch=='A':
   print "Vowel"
elif ch=='e' or ch=='E':
   print "Vowel"
elif ch=='i' or ch=='I':
   print "Vowel"
elif ch=='o' or ch=='O':
   print "Vowel"
elif ch=='u' or ch=='U':
   print "Vowel"
else:
   print "Consonant"
	Vowel

Write a Python program to find the grade of a Student for the marks secured in 5 subjects.

grade.py	Output

#read marks for 5 subjects
total=0
s1=input("Enter marks for s1:")
s2=input("Enter marks for s2:")
s3=input("Enter marks for s3:")
s4=input("Enter marks for s4:")
s5=input("Enter marks for s5:")
#find the total
total=(s1+s2+s3+s4+s5)
print "The Total is :",total
#find the avg
avg=total/5
if avg>90 and avg<100:
   print "Grade is A+"
elif avg>80 and avg <90:
   print "Grade is A"
elif avg>70 and avg <80:
   print "Grade is B+"
elif avg>60 and avg <70:
   print "Grade is B"
elif avg>50 and avg <60:
	Enter marks for s1:78
	Enter marks for s2:90
	Enter marks for s3:96
	Enter marks for s4:98
	Enter marks for s5:93
	The Total is : 455
	Grade is A+

print "Grade is C" else: print "Grade is D"	

Loop Statements

The loops are used to execute some finite number of statements in block repeatedly until some condition is satisfied. There are two loop statements in Python, for and while.

for statement

For loops iterate over a given sequence or list. It is helpful in running a loop on each item in the list. The general form of "for" loop in Python will be as follow:

```
for variable in [value1, value2, etc.]: # list
        statement1
        statement2
        …………
        Statement N
```

Here variable is the name of the variable. And **in** is the keyword. Inside the square brackets a sequence of values are separated by comma. In Python, a comma-separated sequence of data items that are enclosed in a set of square brackets is called a ***list***. The list is created with help of [] square brackets. The list also can be created with help of tuple. We can also use **range()** function to create the list. The general form of the range() function will be as follow:

- **range(number)** –ex: range(10) –It takes all the values from 0 to 9
- **range(start,stop, interval_size)** –ex: range(2,10,2)-It lists all the numbers such as 2,4,6,8.
- **Range(start,stop)**-ex: range(1,6), lists all the numbers from 1to 5, but not 6. Here, by default the interval size is 1.

Write a Python Program to find the sum of all the items in the list using for loop.

fortest.py	Output

#sum of all items in the list
s=0
for x in [1,2,3,4,5]: **# list**
   s=s+x
print "The sum of all items in the list is:",s
	The sum of all items in the list is: 15

Write a Python Program to find the square of all the items in the list using for loop.

squaretest.py	Output

#square of all items in the list print "The Number Square" print " ------------------" for x inrange(1,6): **# list** print "The square of ",x,"is ",x**2 print " -----------------"	The Number Square ------------------- The square of 1 is 1 The square of 2 is 4 The square of 3 is 9 The square of 4 is 16 The square of 5 is 25 -----------------

while statement

While loops repeat as long as a certain boolean condition is met. The block of statements is repeatedly executed as long as the condition is evaluated to True. The general form of while will be as follow:

```
while condition:
        statement1
        statement2
        ………….
        ststementN
```

Write a Python Program to the sum of first N natural numbers using the while loop.

whiletest.py	Output

#sum of first natural numbers using while
s=0
n=input("Enter any number :")
while(n>0):
   s=s+n
   n=n-1
print "The sum is :",s
	Enter any number :10
	The sum is : 55

ELSE for a loop

- Loop statements may have an else clause
- It is executed when the loop terminates through exhaustion of the list (with for loop)
- It is executed when the condition becomes false (with while loop) ,But not when the loop is terminated by a break statement
- Example: printing all primes numbers up to 1000

```
for n in range(2,1000):
for x in range(2, n):
      if (n%x)==0:
      break
else:
      print(n)
```

Jump Statements: we have three jump statements: break, continue and pass.

break statement

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

Syntax

The syntax for a **break** statement in Python is as follows −

```
break
```

Example Program:

#first Example

```
for letter in ‘Python’:  # First Example
   if letter == ‘h’:
      break
   print ‘Current Letter :’, letter
```

Output

```
Current Letter : P
Current Letter : y
Current Letter : t
```

#second Example

```
var = 10                # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
print "Good bye!"
```

```
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

Continue Statement

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

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

Syntax

```
continue
```

#first example

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print 'Current Letter :', letter
```

Output

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

#second Example

```
var = 10                    # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print 'Current variable value :', var
print "Good bye!"
```

Output

```
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

Pass statement

- The pass statement does nothing
- It can be used when a statement is required syntactically but the program requires no action
- Example: creating an infinite loop that does nothing

```
while True:
        pass
```

********-------End--------of------2nd ------Unit---- *******