

PYTHON PROGRAMMING
[KNC-302]

LECTURE NOTES

B.TECH II YEAR – II SEM
(KNC-302)
(2022-2023)

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

SHEAT ENGINEERING COLLEGE BABATPUR VARANSI

SYLLABUS

OUTCOMES: Upon completion of the course, students will be able to

TEXT BOOKS

1. Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist``, 2nd edition, Updated for Python 3, Shroff/O'Reilly Publishers, 2016.
2. R. Nageswara Rao, "Core Python Programming", dreamtech
3. Python Programming: A Modern Approach, Vamsi Kurama, Pearson

REFERENCE BOOKS:

1. Core Python Programming, W.Chun, Pearson.
2. Introduction to Python, Kenneth A. Lambert, Cengage
3. Learning Python, Mark Lutz, Orielly

INDEX

UNIT	TOPIC	PAGE NO
I		
II		
III		

INTRODUCTION DATA, EXPRESSIONS, STATEMENTS

Introduction to Python and installation, data types: Int, float, Boolean, string, and list; variables, expressions, statements, precedence of operators, comments; modules, functions--
- function and its use, flow of execution, parameters and arguments.

Introduction to Python and installation:

Python is a widely used general-purpose, high level programming language. It was initially designed by **Guido van Rossum in 1991** and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions- **Python 2 and Python 3**.

- On 16 October 2000, Python 2.0 was released with many new features.
- On 3rd December 2008, Python 3.0 was released with more testing and includes new features.

Beginning with Python programming:

1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like <https://ide.geeksforgeeks.org/>, <http://ideone.com/> or <http://codepad.org/> that can be used to start Python without installing an interpreter.

Windows: There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) which is installed when you install the python software from <http://python.org/downloads/>

2) Writing first program:

Script Begins

Statement1

Statement2

Statement3

Script Ends

Differences between scripting language and programming language:

SCRIPTING LANGUAGE	PROGRAMMING LANGUAGE
A programming language that supports scripts: programs written for a special run-time environment that automate the execution of tasks	A formal language, which comprises a set of instructions used to produce various kinds of output
Execution speed is slow	Compiler-based languages are executed much faster while interpreter-based languages are executed slower
Can be divided into client-side scripting languages and server-side scripting languages	Can be divided into high-level, low-level languages or compiler-based or interpreter-based languages
Easier to learn	Not as easy to learn
Ex: JavaScript, Perl, PHP, Python and Ruby	Ex: C, C++, and Assembly
Mostly used for web development	Used to develop various applications such as desktop, web, mobile, etc.

Why to use Python:

The following are the primary factors to use python in day-to-day life:

1. Python is object-oriented

Structure supports such concepts as polymorphism, operation overloading and multiple inheritance.

2. Indentation

Indentation is one of the greatest feature in python

3. It's free (open source)

Downloading python and installing python is free and easy

4. It's Powerful

- Dynamic typing
- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, sciPy)
- Automatic memory management

5. It's Portable

- Python runs virtually every major platform used today
- As long as you have a compatible python interpreter installed, python programs will run in exactly the same manner, irrespective of platform.

6. It's easy to use and learn

- No intermediate compile
- Python Programs are compiled automatically to an intermediate form called byte code, which the interpreter then reads.
- This gives python the development speed of an interpreter without the performance loss inherent in purely interpreted languages.
- Structure and syntax are pretty intuitive and easy to grasp.

7. Interpreted Language

Python is processed at runtime by python Interpreter

8. Interactive Programming Language

Users can interact with the python interpreter directly for writing the programs

9. Straight forward syntax

The formation of python syntax is simple and straight forward which also makes it popular.

Installation:

There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) which is installed when you install the python software from <http://python.org/downloads/>

Steps to be followed and remembered:

Step 1: Select Version of Python to Install.

Step 2: Download Python Executable Installer.

Step 3: Run Executable Installer.

Step 4: Verify Python Was Installed On Windows.

Step 5: Verify Pip Was Installed.

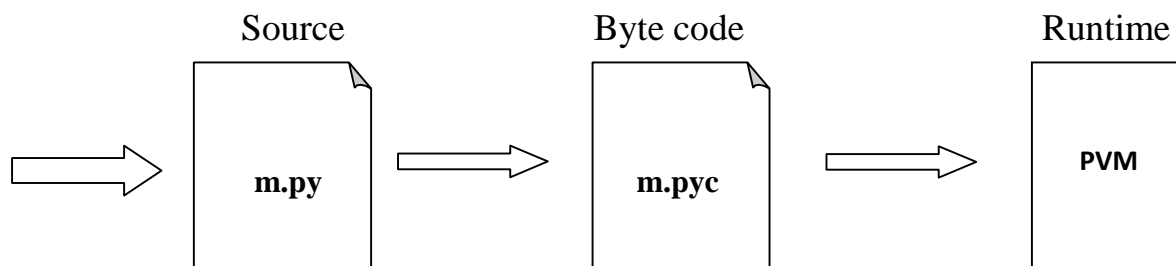
Step 6: Add Python Path to Environment Variables (Optional)



Working with Python

Python Code Execution:

Python's traditional runtime execution model: Source code you type is translated to byte code, which is then run by the Python Virtual Machine (PVM). Your code is automatically compiled, but then it is interpreted.



Source code extension is .py
Byte code extension is .pyc (Compiled python code)

There are two modes for using the Python interpreter:

- Interactive Mode
- Script Mode

Running Python in interactive mode:

Without passing python script file to the interpreter, directly execute code to Python prompt. Once you're inside the python interpreter, then you can start.

```
>>> print("hello world")
```

```
hello world
```

Relevant output is displayed on subsequent lines without the >>> symbol

```
>>> x=[0,1,2]
```

Quantities stored in memory are not displayed by default.

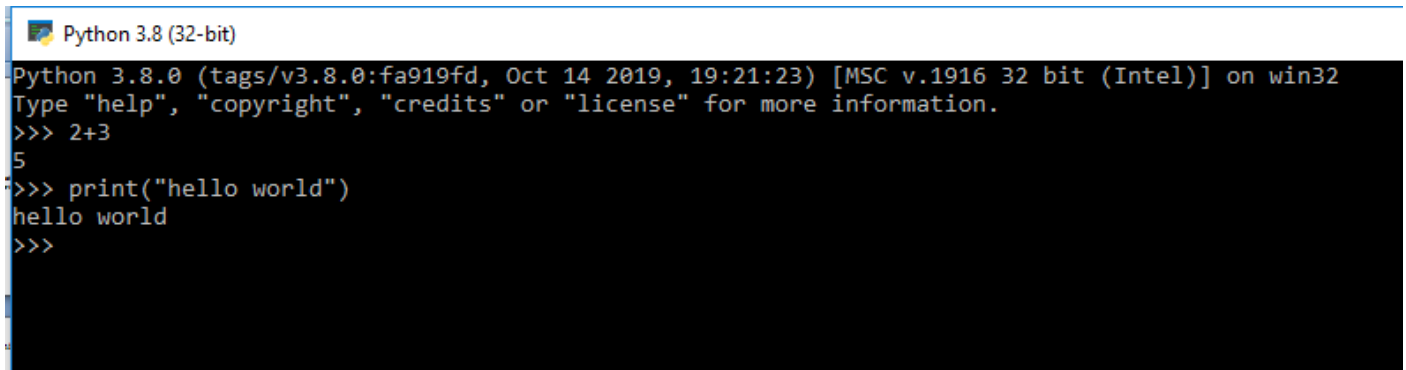
```
>>> x
```

#If a quantity is stored in memory, typing its name will display it.

```
[0, 1, 2]
```

```
>>> 2+3
```

```
5
```

A screenshot of a Python 3.8 (32-bit) interpreter window. The title bar reads "Python 3.8 (32-bit)". The window content shows the Python prompt and the following interactions:

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> print("hello world")
hello world
>>>
```

The chevron at the beginning of the 1st line, i.e., the symbol >>> is a prompt the python interpreter uses to indicate that it is ready. If the programmer types 2+6, the interpreter replies 8.

Running Python in script mode:

Alternatively, programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file. To execute the script by the interpreter, you have to tell the interpreter the name of the file. For example, if you have a script name MyFile.py and you're working on Unix, to run the script you have to type:

python MyFile.py

Working with the interactive mode is better when Python programmers deal with small pieces of code as you can type and execute them immediately, but when the code is more than 2-4 lines, using the script for coding can help to modify and use the code in future.

Example:

```
C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\pyyy>python e1.py
resource open
the no cant be divisible zero division by zero
resource close
finished
```

Data types:

The data stored in memory can be of many types. For example, a student roll number is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Int:

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
>>> print(24656354687654+2)
```

```
24656354687656
```

```
>>> print(20)
```

```
20
```

```
>>> print(0b10)
```

```
2
```

```
>>> print(0B10)
```

```
2
```

```
>>> print(0X20)
```

```
32
```

```
>>> 20
```

```
20
```

```
>>> 0b10
```

```
2
```

```
>>> a=10
```

```
>>> print(a)
```

```
10
```

To verify the type of any object in Python, use the type() function:

```
>>> type(10)
```

```
<class 'int'>
```

```
>>> a=11
```

```
>>> print(type(a))
```

```
<class 'int'>
```

Float:

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
>>> y=2.8
```

```
>>> y
```

```
2.8
```

```
>>> y=2.8
```

```
>>> print(type(y))
```

```
<class 'float'>
```

```
>>> type(.4)
```

```
<class 'float'>
```

```
>>> 2.
```

2.0

Example:`x = 35e3``y = 12E4``z = -87.7e100``print(type(x))``print(type(y))``print(type(z))`**Output:**`<class 'float'>``<class 'float'>``<class 'float'>`**Boolean:**

Objects of Boolean type may have one of two values, True or False:

`>>> type(True)``<class 'bool'>``>>> type(False)``<class 'bool'>`**String:**

1. Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

- 'hello' is the same as "hello".
- Strings can be output to screen using the print function. **For example: print("hello").**

`>>> print("mrcet college")``mrcet college``>>> type("mrcet college")``<class 'str'>`

```
>>> print('mrcet college')
```

```
mrcet college
```

```
>>> " "
```

```
' '
```

If you want to include either type of quote character within the string, the simplest way is to delimit the string with the other type. If a string is to contain a single quote, delimit it with double quotes and vice versa:

```
>>> print("mrcet is an autonomous (' ) college")
```

```
mrcet is an autonomous (' ) college
```

```
>>> print('mrcet is an autonomous (" ) college')
```

```
mrcet is an autonomous (" ) college
```

Suppressing Special Character:

Specifying a backslash (\) in front of the quote character in a string “escapes” it and causes Python to suppress its usual special meaning. It is then interpreted simply as a literal single quote character:

```
>>> print("mrcet is an autonomous (\') college")
```

```
mrcet is an autonomous (' ) college
```

```
>>> print('mrcet is an autonomous (\") college')
```

```
mrcet is an autonomous (" ) college
```

The following is a table of escape sequences which cause Python to suppress the usual special interpretation of a character in a string:

```
>>> print('a\
```

```
....b')
```

```
a. ..b
```

```
>>> print('a\
```

```
b\
```

```
c')
```

```
abc
```

```
>>> print('a \n b')
```

```
a
```

```
b
```

```
>>> print("mrcet \n college")
```

```
mrcet
```

```
college
```

Escape Sequence	Usual Interpretation of Character(s) After Backslash	“Escaped” Interpretation
\'	Terminates string with single quote opening delimiter	Literal single quote (') character
\"	Terminates string with double quote opening delimiter	Literal double quote (") character
\newline	Terminates input line	Newline is ignored
\\	Introduces escape sequence	Literal backslash (\) character

In Python (and almost all other common computer languages), a tab character can be specified by the escape sequence `\t`:

```
>>> print("a\tb")
```

```
a      b
```

List:

- It is a general purpose most widely used in data structures
- List is a collection which is ordered and changeable and allows duplicate members. (Grow and shrink as needed, sequence type, sortable).
- To use a list, you must declare it first. Do this using square brackets and separate values with commas.
- We can construct / create list in many ways.

Ex:

```
>>> list1=[1,2,3,'A','B',7,8,[10,11]]
```

```
>>> print(list1)
```

```
[1, 2, 3, 'A', 'B', 7, 8, [10, 11]]
```

```
-----  
>>> x=list()  
  
>>> x  
  
[]  
  
-----  
  
>>> tuple1=(1,2,3,4)  
>>> x=list(tuple1)  
  
>>> x  
  
[1, 2, 3, 4]
```

Variables:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example –

```
a= 100      # An integer assignment
```

```
b = 1000.0   # A floating point
```

```
c = "John"   # A string
```

```
print (a)
```

```
print (b)
```

```
print (c)
```

This produces the following result –

```
100
```

```
1000.0
```

```
John
```

Multiple Assignment:

Python allows you to assign a single value to several variables simultaneously.

For example :

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

For example –

```
a,b,c = 1,2,"mrcet"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Output Variables:

The Python print statement is often used to output variables.

Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 5          # x is of type int
x = "mrcet "   # x is now of type str
print(x)
```

Output: mrcet

To combine both text and a variable, Python uses the “+” character:

Example

```
x = "awesome"
print("Python is " + x)
```

Output

Python is awesome

You can also use the + character to add a variable to another variable:

Example

```
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

Output:

Python is awesome

Expressions:

An expression is a combination of values, variables, and operators. An expression is evaluated using assignment operator.

Examples: $Y = x + 17$

```
>>> x=10
```

```
>>> z=x+20
```

```
>>> z
```

```
30
```

```
>>> x=10
>>> y=20
>>> c=x+y
>>> c
30
```

A value all by itself is a simple expression, and so is a variable.

```
>>> y=20
>>> y
20
```

Python also defines expressions only contain identifiers, literals, and operators. So,

Identifiers: Any name that is used to define a class, function, variable module, or object is an identifier.

Literals: These are language-independent terms in Python and should exist independently in any programming language. In Python, there are the string literals, byte literals, integer literals, floating point literals, and imaginary literals.

Operators: In Python you can implement the following operations using the corresponding tokens.

Operator	Token
add	+
subtract	-
multiply	*
Integer Division	/
remainder	%
Binary left shift	<<
Binary right shift	>>
and	&
or	\
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Check equality	==
Check not equal	!=

Some of the python expressions are:**Generator expression:****Syntax:** (compute(var) for var in iterable)

```
>>> x = (i for i in 'abc') #tuple comprehension
>>> x
<generator object <genexpr> at 0x033EEC30>

>>> print(x)
<generator object <genexpr> at 0x033EEC30>
```

You might expect this to print as ('a', 'b', 'c') but it prints as <generator object <genexpr> at 0x02AAD710> The result of a tuple comprehension is not a tuple: it is actually a generator. The only thing that you need to know now about a generator now is that you can iterate over it, but ONLY ONCE.

Conditional expression:**Syntax:** true_value if Condition else false_value

```
>>> x = "1" if True else "2"

>>> x

'1'
```

Statements:

A statement is an instruction that the Python interpreter can execute. We have normally two basic statements, the assignment statement and the print statement. Some other kinds of statements that are if statements, while statements, and for statements generally called as control flows.

Examples:

An assignment statement creates new variables and gives them values:

```
>>> x=10
```

```
>>> college="mrcet"
```

An print statement is something which is an input from the user, to be printed / displayed on to the screen (or) monitor.

```
>>> print("mrcet colege")
```

```
mrcet college
```

Precedence of Operators:

Operator precedence affects how an expression is evaluated.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first multiplies $3*2$ and then adds into 7.

Example 1:

```
>>> 3+4*2
```

```
11
```

Multiplication gets evaluated before the addition operation

```
>>> (10+10)*2
```

```
40
```

Parentheses () overriding the precedence of the arithmetic operators

Example 2:

```
a = 20
```

```
b = 10
```

```
c = 15
```

```
d = 5
```

```
e = 0
```

```
e = (a + b) * c / d    #( 30 * 15 ) / 5  
print("Value of (a + b) * c / d is ", e)
```

```
e = ((a + b) * c) / d    # (30 * 15 ) / 5  
print("Value of ((a + b) * c) / d is ", e)
```

```
e = (a + b) * (c / d);    # (30) * (15/5)
```

```
print("Value of (a + b) * (c / d) is ", e)
```

```
e = a + (b * c) /  
d;
```

```
# 20 + (150/5)  
print("Value of a +  
(b * c) / d is ", e)
```

Output:


```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-  
32/pyyy/opprec.py Value of (a + b) * c / d is 90.0  
Value of ((a + b)  
* c) / d is 90.0  
Value of (a + b)  
* (c / d) is 90.0  
Value of a + (b *  
c) / d is 50.0
```

Comments:

Single-line comments begins with a hash(#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of line.

A Multi line comment is useful when we need to comment on many lines. In python, triple double quote(“ “ “) and single quote(‘ ‘ ‘) are used for multi-line commenting.

Example:

 comm.py - C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/comm.py...

File Edit Format Run Options Window Help

```
# Write a python program to add numbers  
a=10 #assigning value to variable a  
b=20 #assigning value to variable b  
""" print the value of a and b using a new variable """  
''' print the value of a and b using a new variable '''  
c=a+b  
print(c)  
|
```

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/comm.py

Python Literals

Python Literals can be defined as data that is given in a variable or constant.

Python supports the following literals:

1. String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes to create a string.

Example:

```
Backward Skip 10sPlay VideoForward Skip 10s
```

1. "Aman" , '12345'

Types of Strings:

There are two types of Strings supported in Python:

a) Single-line String- Strings that are terminated within a single-line are known as Single line Strings.

Example:

1. text1='hello'

b) Multi-line String - A piece of text that is written in multiple lines is known as multiple lines string.

There are two ways to create multiline strings:

1) Adding black slash at the end of each line.

Example:

1. text1='hello\


```
2. user'  
3. print(text1)  
   'hellouser'
```

2) Using triple quotation marks:-

Example:

```
1. str2="""welcome  
2. to  
3. SSSIT"  
4. print str2
```

Output:

```
welcome  
to  
SSSIT
```

II. Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+bj where a forms the real part and b forms the imaginary part of the complex number. eg: 3.14j

Example - Numeric Literals

```
1. x = 0b10100 #Binary Literals  
2. y = 100 #Decimal Literal  
3. z = 0o215 #Octal Literal  
4. u = 0x12d #Hexadecimal Literal
```

- 5.
6. **#Float Literal**
7. float_1 = 100.5
8. float_2 = 1.5e2
- 9.
10. **#Complex Literal**
11. a = 5+3.14j
- 12.
13. **print**(x, y, z, u)
14. **print**(float_1, float_2)
15. **print**(a, a.imag, a.real)

Output:

```
20 100 141 301
100.5 150.0
(5+3.14j) 3.14 5.0
```

III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

Example - Boolean Literals

1. x = (1 == True)
2. y = (2 == False)
3. z = (3 == True)
4. a = True + 10
5. b = False + 10
- 6.
7. **print**("x is", x)
8. **print**("y is", y)
9. **print**("z is", z)
10. **print**("a:", a)
11. **print**("b:", b)

Output:

```
x is True
```

```
y is False  
z is False  
a: 11  
b: 10
```

IV. Special literals.

Python contains one special literal i.e., **None**.

None is used to specify to that field that is not created. It is also used for the end of lists in Python.

Example - Special Literals

1. val1=10
2. val2=None
3. **print**(val1)
4. **print**(val2)

Output:

```
10  
None
```

V. Literal Collections.

Python provides the four types of literal collection such as List literals, Tuple literals, Dict literals, and Set literals.

List:

- List contains items of different data types. Lists are mutable i.e., modifiable.
- The values stored in List are separated by comma(,) and enclosed within square brackets([]). We can store different types of data in a List.

Example - List literals

1. list=['John',678,20.4,'Peter']
2. list1=[456,'Andrew']
3. **print**(list)
4. **print**(list + list1)

Output:

```
['John', 678, 20.4, 'Peter']  
['John', 678, 20.4, 'Peter', 456, 'Andrew']
```

Dictionary:

- Python dictionary stores the data in the key-value pair.
- It is enclosed by curly-braces {} and each pair is separated by the commas(,).

Example

1. dict = {'name': 'Pater', 'Age':18,'Roll_nu':101}
2. **print**(dict)

Output:

```
{'name': 'Pater', 'Age': 18, 'Roll nu': 101}
```

Tuple:

- Python tuple is a collection of different data-type. It is immutable which means it cannot be modified after creation.
- It is enclosed by the parentheses () and each element is separated by the comma(,).

Example

1. tup = (10,20,"Dev",[2,3,4])
2. **print**(tup)

Output:

```
(10, 20, 'Dev', [2, 3, 4])
```

Set:

- Python set is the collection of the unordered dataset.
- It is enclosed by the {} and each element is separated by the comma(,).

Example: - Set Literals

1. set = {'apple','grapes','guava','papaya'}

2. `print(set)`

Output:

```
{'guava', 'apple', 'papaya', 'grapes'}
```