

PYTHON PROGRAMMING (20MCA21C)

UNIT - I

Introduction to Python

FACULTY:



Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,



Associate Professor and Head,
Post Graduate and Research Department of Computer Applications,
Government Arts College (Autonomous), Coimbatore - 641 018.



PYTHON PROGRAMMING (20MCA21C) SYLLABUS

- **UNIT I: Introduction to Python:** Python Overview - Getting Started with Python - Python Identifiers - Reserved Keywords - Variables - Standard Data Types - Operators. Statement and Expression - String Operations - Boolean Expressions - Control Statements - Iteration - **while** Statement - Input.
- **UNIT II: Functions:** Introduction - Built-in Functions - Composition of Functions - User Defined Functions - Parameters and Arguments - Function Calls - The **return** Statement - Python Recursive Function - The Anonymous Functions - Writing Python Scripts.

- 
- 
- **Unit III: Strings:** Strings - Compound data types - `len()` function - String slices - String traversal - String formatting operators and functions. **Lists:** Values and accessing elements - lists are mutable - Traversing and deleting elements - Built-in operators and methods.
 - **Unit IV: Tuples:** Creating tuples-accessing values - tuples assignment - tuples as return values - variable length argument tuples - basic tuple operations - built-in tuple functions. **Dictionaries:** Creating and accessing a dictionary - updating and deleting - properties of dictionary keys - operations and built-in dictionary methods. **Exceptions:** Exceptions with Arguments - User-Defined Exceptions.



□ **Unit V: Classes and Objects:** Overview of OOP
(Object-Oriented Programming)- Class Definitions
Creating Objects-Objects as Arguments - Objects as
Return Values - Built-in Class Attributes -Inheritance -
Method Overloading.

□ **TEXT BOOKS:**

□ E. Balagurusamy, “Introduction To Computing And
Problem Solving Using Python”, McGraw Hill Education
Private Limited, New Delhi.

□ **REFERENCE BOOKS:**

□ Mark Lutz, David Ascher, “Learning Python”, Shroff
Publishers & Distributors Private Limited,2009.



History of Python



- Created in 1989 by Guido van Rossum
 - Created as a scripting language for administrative tasks
 - Based on All Basic Code (ABC) and Modula-3
 - Added extensibility
 - Named after comic troupe Monty Python
- Released publicly in 1991
 - Growing community of Python developers
 - Evolved into well-supported programming language



History of Python



- Modules
 - Reusable pieces of software
 - Can be written by any Python developer
 - Extend Python's capabilities
- Python Web site at www.python.org
 - Primary distribution center for Python source code, modules and documentation




History of Python

- Python
 - Designed to be portable and extensible
 - Originally implemented on UNIX
 - Programs often can be ported from one operating system to another without any change
 - Syntax and design promote good programming practices with rapid development times
 - Simple enough to be used by beginning programmers
 - Powerful enough to attract professionals



World-Class Software Companies That Use Python

- Google.
 - Facebook.
 - Instagram.
 - Spotify.
 - Quora.
 - Netflix.
 - Dropbox.
- 



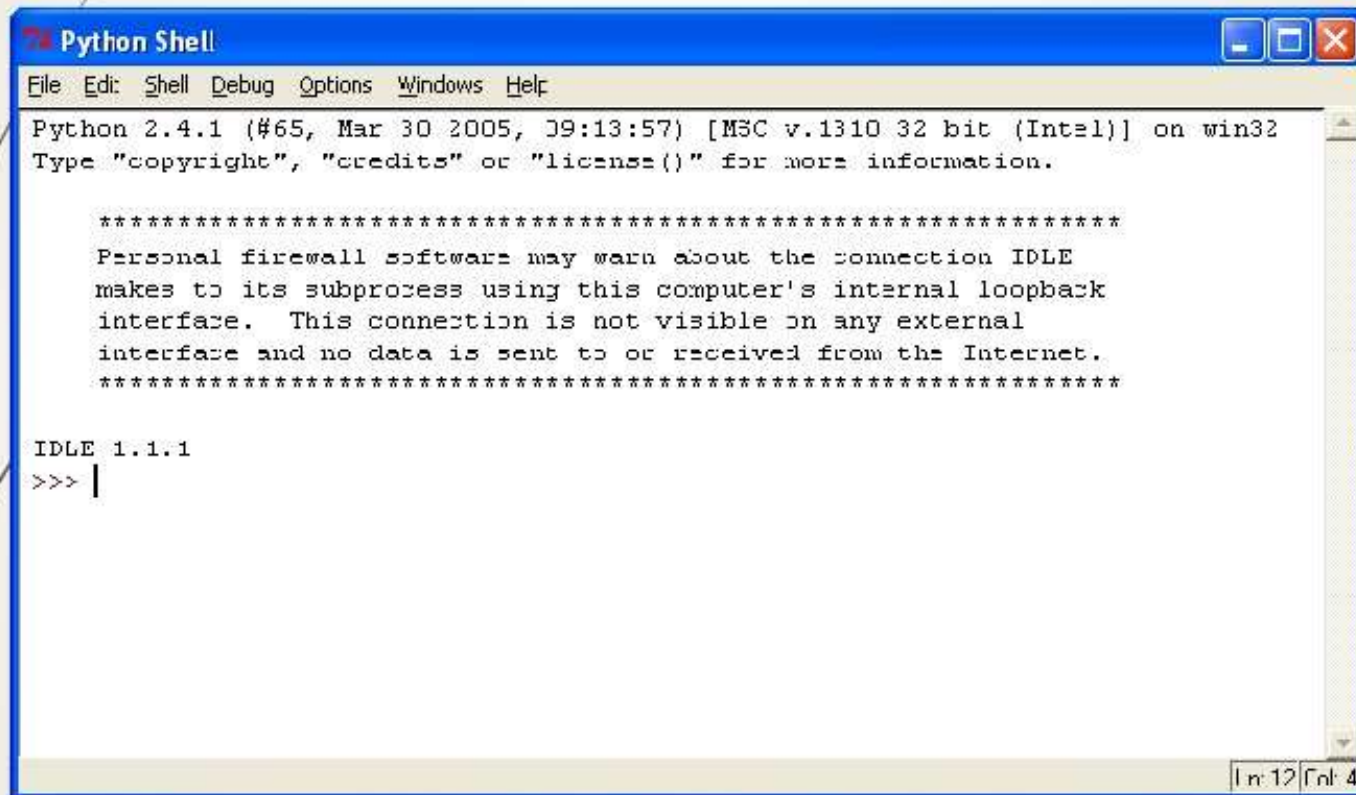
Getting Started with Python



Setting Up Python on Windows

- Go to <http://www.python.org> and get the latest distribution (3.8)
 - Online tutorials
 - Python related websites
- Use the distribution on the CD ROM supplied with the textbook
 - Examples from the book
- Use all the defaults when installing

Python IDLE



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.1.1
>>> |
Ln: 12 | Col: 4
```



First Python Program



- At the prompt (>>>) type:
 - `print "MCA AT GAC"`
 - Press [Enter]
 - `print "COIMBATORE"`
 - Press [Enter]
- Programming in Python
- Interactive mode gives you immediate feedback
- Not designed to create programs to save and run later
- Script Mode
 - Write, edit, save, and run (later)
 - Word processor for your code
- Save your file using the “.py” extension



Program Documentation



- Comment lines provide documentation about your program
 - Anything after the “#” symbol is a comment
 - Ignored by the computer

- # I AM A Programmer
- # First Python Program
- # MCA ,GAC

Expressions

- Expression: A data value or set of operations to compute a value.
 - Examples: $1 + 4 * 3$
 - 42
- Arithmetic operators we will use:
 - $+ - * /$ addition, subtraction/negation, multiplication, division
 - $\%$ modulus, remainder
 - $**$ exponentiation
- precedence: Order in which operations are computed.
 - $* / \% **$ have a higher precedence than $+ -$
 $1 + 3 * 4$ is 13
 - Parentheses can be used to force a certain order of evaluation.
 $(1 + 3) * 4$ is 16

Python has useful commands for performing calculations.

Command name	Description
<code>abs(value)</code>	absolute value
<code>ceil(value)</code>	rounds up
<code>cos(value)</code>	cosine, in radians
<code>floor(value)</code>	rounds down
<code>log(value)</code>	logarithm, base e
<code>log10(value)</code>	logarithm, base 10
<code>max(value1, value2)</code>	larger of two values
<code>min(value1, value2)</code>	smaller of two values
<code>round(value)</code>	nearest whole number
<code>sin(value)</code>	sine, in radians
<code>sqrt(value)</code>	square root

commands, you must write the

□ Variable: A named piece of memory that can store a value.

□ Usage:

- Compute an expression's result,
- store that result into a variable,
- and use that variable later in the program.

□ Assignment statement: Stores a value into a variable.

□ Syntax:

□ `name = value`

□ Examples: `a = 15`

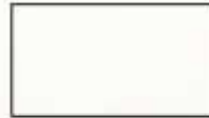
□ `pi = 3.14`

□ `a pi`

□ `15 3.14`


□ A variable that has been given a value can be used in expressions.

□ `x + 5 is 20`





Standard Data Types

- 
- 1.Numeric
 - 2. String
 - 3.Lists
 - 4.Tuple
 - 5.Dictionary
 - 6.Boolean
 - 7.Sets



1.Numeric:

- Integers and floating point values are Numeric.
- Examples
- `>>> num1=34`
- `>>>num2=6.89`
- `>>>num1`
- 34
- `>>num2`
- 6.89



2.String

- Single quotes or double quotes can be used to represent strings.
- `>>>str1='MCA'`
- `>>>str1`
- MCA

- `>>>str1 + 'GAC, Coimbatore'`
- `>>>MCA GAC, Coimbatore`



Lists



- A list is an ordered and indexable sequence and contain different types of items
- `>>>mylist=[1,"two",3.0]`
- `>>>mylist`
- `[1,"two",3.0]`



Tuples



- A Tuple is used to store sequence of items enclosed in paranthesis.
- Examples
- `>>>Tup1={3,"three",9.7}`
- `>>>Tup1`
- `{3,"three",9.7}`



Dictionary



- Python dictionary is a unordered collection of key-value pairs.
- Examples
- `>>>Dict1={1:"first","second":2}`
- `>>>Dict1`
- `{1:"first","second":2}`



Boolean


- Boolean data type stores TRUE or FALSE values only.
- Example
- `>>>A=True`
- `>>>type(A)`
- `<type 'bool'>`





Sets



- A set is an unordered collection of data .Sets do not contain any duplicate values or elements
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.
- Example
 - `>>>Set1=set([1,2,3,4])`
 - `>>>set1`
 - `([1,2,3,4])`

- 
- **print** : Produces text output on the console.
 - **Syntax:**
 - `print "Message"`
 - `print Expression`
 - Prints the given text message or expression value on the console, and moves the cursor down to the next line.
 - `print Item1, Item2, ..., ItemN`
 - Prints several messages and/or expressions on the same line.
 - **Examples:**
 - `print " GAC, CBE!"`
 - `age = 10`
 - `print "You have", 20 - age, "years for 20"`
 - **Output:**
 - `GAC, CBE!`
 - `You have 10 years for 20`

- 
- 
- input : Reads a number from user input.
 - You can assign (store) the result of input into a variable.
 - Example:
 - `age = input("How old are you? ")`
 - `print "Your age is", age`
 - Output:
 - How old are you? 23
 - Your age is 23



Operators



- ❑ Python language supports the following types of operators.
- ❑ Arithmetic Operators
- ❑ Comparison (Relational) Operators
- ❑ Assignment Operators
- ❑ Logical Operators
- ❑ Bitwise Operators
- ❑ Membership Operators
- ❑ Identity Operators

Arithmetic operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
** Exponent	Performs exponential (power) calculation on operators	$a^{**}b = 10 \text{ to the power } 20$
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	$9//2 = 4$ and $9.0//2.0 = 4.0$, $-11//3 = -4$, $-11.0//3 = -4.0$

Comparison operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.



Comparison operators



Operator	Description	Example
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	<code>c = a + b</code> assigns value of <code>a + b</code> into <code>c</code>
<code>+=</code> Add AND	It adds right operand to the left operand and assign the result to left operand	<code>c += a</code> is equivalent to <code>c = c + a</code>
<code>-=</code> Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	<code>c -= a</code> is equivalent to <code>c = c - a</code>
<code>*=</code> Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	<code>c *= a</code> is equivalent to <code>c = c * a</code>
<code>/=</code> Divide AND	It divides left operand with the right operand and assign the result to left operand	<code>c /= a</code> is equivalent to <code>c = c / a</code>
<code>%=</code> Modulus AND	It takes modulus using two operands and assign the result to left operand	<code>c %= a</code> is equivalent to <code>c = c % a</code>
<code>**=</code> Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	<code>c **= a</code> is equivalent to <code>c = c ** a</code>
<code>//=</code> Floor Division	It performs floor division on operators and assign value to the left operand	<code>c //= a</code> is equivalent to <code>c = c // a</code>

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False



Repetition (loops) and Selection (if/else)





□ for loop: Repeats a set of statements over a group of values.

□ Syntax:

□ for variableName in groupOfValues:

□ statements

□ We indent the statements to be repeated with tabs or spaces.

□ variableName gives a name to each value, so you can refer to it in the statements.

□ groupOfValues can be a range of integers, specified with the range function.

□ Example:

□ for x in range(1, 6):

□ print x, "squared is", x * x

□ Output:

□ 1 squared is 1

□ 2 squared is 4

□ 3 squared is 9

□ 4 squared is 16

□ 5 squared is 25



- The range function specifies a range of integers:
 - `range(start, stop)` - the integers between start (inclusive) and stop (exclusive)
- It can also accept a third value specifying the change between values.
 - `range(start, stop, step)` - the integers between start (inclusive) and stop (exclusive) by step

□ **Example:**

```
for x in range(5, 0, -1):  
    print x  
print "Blastoff!"
```

□ **Output:**

```
5  
4  
3  
2  
1  
Blastoff!
```



□ Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a cumulative sum.

□ `sum = 0`

□ `for i in range(1, 11):`

□ `sum = sum + (i * i)`

□ `print "sum of first 10 squares is", sum`

□ Output:

□ `sum of first 10 squares is 385`

□ if statement: Executes a group of statements if a certain condition is true. Otherwise, they are skipped.

□ Syntax:

□ if condition:

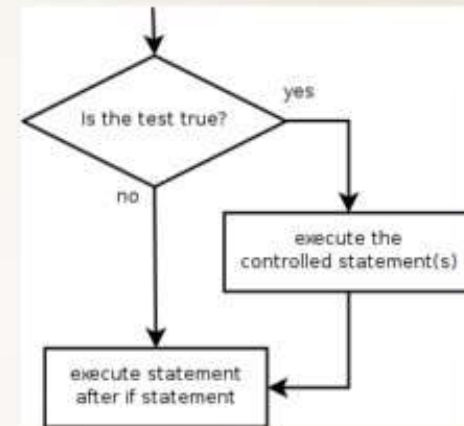
□ statements

□ Example:

□ `gpa = 3.4`

□ `if gpa > 2.0:`

□ `print "GPA is greater than 2."`



- if/else statement: Executes one bloc condition is True, and a second bloc

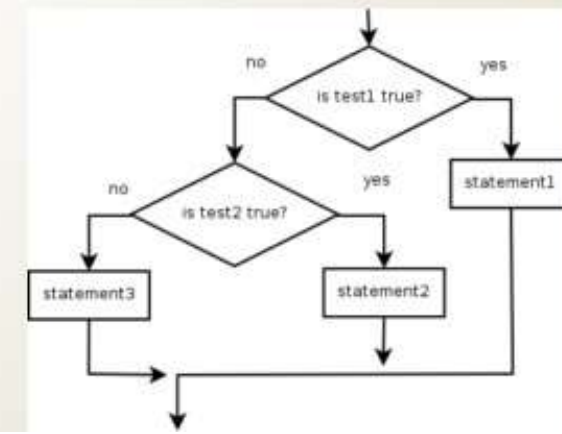
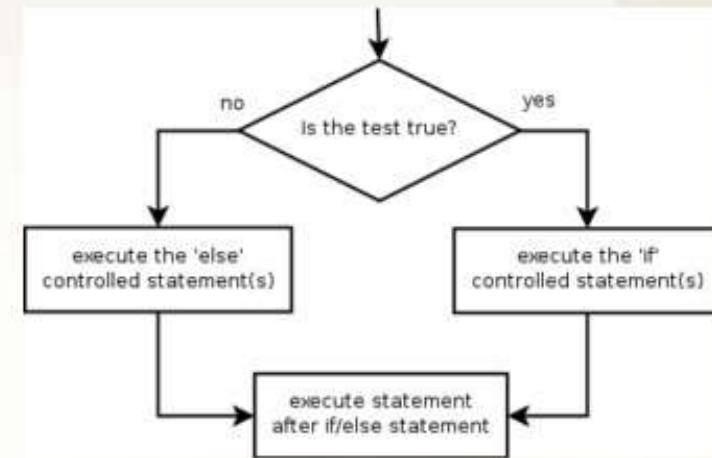
- Syntax:
- if condition:
- statements
- else:
- statements

- Example:

- if gpa > 5.0:
- print "Welcome to Our Univers
- else:
- print "Your application is rejected."

- Multiple conditions can be chained with elif ("else if"):

- if condition:
- statements
- elif condition:
- statements
- else:
- statements



- while loop: Executes a group of statements as long as a condition is True.

- good for indefinite loops (repeat an un

- Syntax:

- while condition:

- statements

- Example:

- number = 1

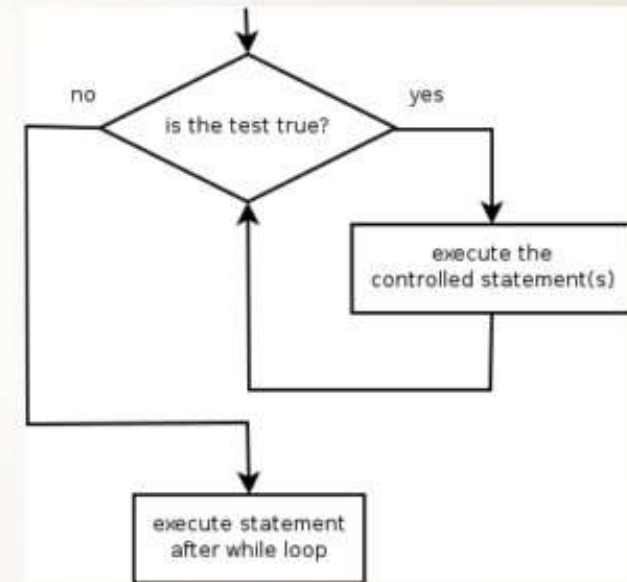
- while number < 100:

- print number

- number = number * 2

- Output:

- 1 2 4 8 16 32 64





□ `raw_input` : Reads a string of text from user input.

□ Example:


□ `name = raw_input(" What's your name? ")`

□ `print name, " is a lovely name!"`

□ Output:

□ What's your name? Shilpa

□ Shilpa is a lovely name!

- 
- text processing: Examining, editing, formatting text.
 - often uses loops that examine the characters of a string one by one
 - A for loop can examine each character in a string in sequence.
 - Example:
 - for c in "GACCBE":
 - print c
 - Output:
 - G
 - A
 - C
 - C
 - B
 - E



Thank you

The Content in this Material are from the Textbooks and Reference books given in the Syllabus