Programming in Python

Course Information

Course Code: KCE-PY

Course Name: Programming in Python

Course Duration: 20 Hrs

Topics covered in “Programming in Python” course

- Day 1
  - Python History & Features
  - Installing Python
  - Basic Syntax
  - Variable and Data Types
  - Operator

- Day 2
  - Conditional Statements
  - Looping
  - Control Statements

Topics covered in “Programming in Python” course

- Day 3
  - Data Structures
  - Strings
  - Tuple
- Day 4
  - Lists
  - Set
  - Dictionary
- Day 5
  - Functions
  - Different types of arguments
  - Recursion

Topics covered in “Programming in Python” course

- Day 6
  - Standard Library
  - Math module
  - List module
  - Date & Time module
- Day 7
  - File Operations
  - Exception handling
- Day 8
  - Introduction to OOC
  - UML

Topics covered in “Programming in Python” course

- Day 9
  - Classes and Objects
  - Methods
  - Inheritance
- Day 10
  - Database Connectivity
  - Executing queries
- Project Development
- KCE Python Certification Exam

Introduction

Why Python for beginners?

- Easy – to - learn
  - Code is 3-5 times shorter than Java
  - 5-10 times shorter than C++
- Stepping Stone to Programming universe
  - Python's methodologies can be used in a broad range of applications
- Bridging the Gap between abstract computing and real world applications
  - Python is used as main programming language to do projects using Raspberry Pi
- Rising Demand for Python Programmers
  - Google, Nokia, Disney, Yahoo, IBM use Python
- Open- Source, Object – Oriented, procedural and functional
  - Not only a Scripting language, also supports Web Development and Database Connectivity

Python v/s C

A simple Program to print “Hello World”

C Code	Python Code

#include<stdio.h>
#include<conio.h>
void main()
{
printf (“Hello World!”)
}
	print (“Hello World!”)

Worldwide Python Users

- **Web Development**
  - Yahoo Groups, Google, Shopzilla
- **Games**
  - Battelefield2, The Temple of Elemental Evil, Vampire
- **Graphics**
  - Walt Disney feature Animation, Blender 3D
- **Science**
  - National Weather Service
  - NASA
  - Environmental Systems Research Institute

Evolution of Python

- **Guido Van Rossum** developed Python in **early 1990s** at National Research Institute for Mathematics and Computer Science, Netherlands.
- Named after a **circus show Monty Python show.**
- Derives its features from many languages like **Java, C++, ABC, C, Modula-3, Smalltalk, Algol-68, Unix shell** and other scripting languages.
- Available under the **GNU General Public License (GPL) – Free and open-source software.**

Python Versions

- **Python v0.9.0 - February, 1991**
  - Features: **Exception Handling, Functions and core data types like List, Dictionary, String** and others. It was object oriented and had module system
- **Python v1.0 - January 1994**
  - Features: **Functional Programming tools lambda, map, filter and reduce**
- **Python v2.0 - October 2000**
  - Features: **List comprehensions, Garbage Collector and support for Unicode.**
- **Python v3.0 - 2008**
  - Known as “Python 3000” and “Py3k**”. It is not backward compatible with v2.0** and its other variants. **Emphasizes more on removal of duplicate programming constructs** and modules

Python Features

- Python is a **High - Level, Interpreted, Interactive and Object - Oriented Programming Language**
- Features include:
  - Beginners Language
  - Extensive Standard Library
  - Cross Platform Compatibility
  - Interactive Mode
  - Portable and Extendable
  - Databases and GUI Programming
  - Scalable and Dynamic Semantics
  - Automatic Garbage Collection

Configuration

- Download and Install Python 3.5: https://www.python.org/downloads/
- Download PyDev_3.8.0 or higher version: http://www.pydev.org/download.html
- Download Eclipse Juno or higher version: https://www.eclipse.org/downloads/index.php
- Install Python on your machine

Note: You need eclipse locally installed in your machine

Procedure:

- Copy paste the contents of the plugin folder of PyDev into plugin folder of Eclipse
- In Eclipse open PyDev perspective (Window -> Open Perspective -> Other -> Pydev)
- Create a PyDev Project
- Select Grammar as 3.0
- Configure interpreter by choosing the .exe file of Python installed in your machine

Day – 1

Python Basics

Commenting Style in Python!

Types of Comments:

- A single line comment starts with hash symbol '#' and end as the line ends.
  - These lines are never executed and are ignored by the interpreter.
    - Single # This is a single line comment

- Multi-line comments starts and ends with triple single quotes '' or triple """ double quotes
  - Used for documentation
    - Triple '' or """

```
"""
Contents here can be used
for documentation
"""
''
An example for multi-line
comments with single quotes
''
```

Multiline statements

- Python statements always end up with a new line, but it also allows multiline statement using “\” character at the end of line as shown below:

```
result = (8+5)*\
         2+\
         9/5
```

- Statements which have (), [], {} brackets and comma, do not need any multiline character to go to next line.

```
customer_details = [101, ‘kevin’,
                    ‘165’, 498.24]
```

Print Statement

- Displays the output on the screen of user
- Python has its own String Format Operator as %

```
print('The value of PI is %5.3f' % 3.1417)
print('The value of PI is approximately %5.2f.' % 3.1417)
print( "Latest Python Version is: %d" % 3.5)
print ("%20s : %d" % ('Python', 3000.34 ))
```

Output:

```
The value of PI is 3.142
The value of PI is approximately 3.14.
Latest Python Version is: 3
Python : 3000
```

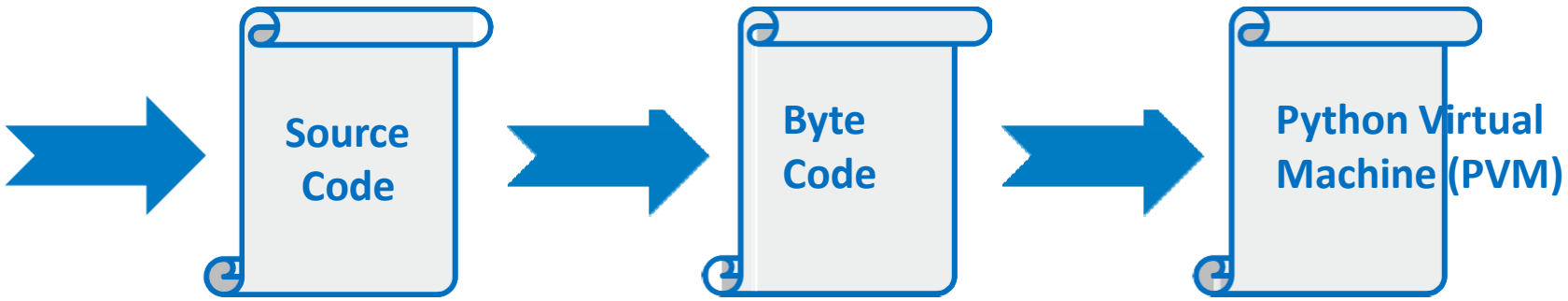
User Input in Python

```
name = input("Enter your name")
print("Welcome to session on Programming in Python,", name)
```

Execute a Python Script

- Execution of python program means execution of the byte code on Python Virtual Machine

```
print(“Hello World!”)                    #prints Hello World!
print("My first sample python script")   #prints My first sample python script
```



Python Data Variables

Programming Constructs in Python

- Given a real world problem, to solve the problem using a program, we need:
  - Logic
  - High level programming language
    - Programming Fundamentals
      - Identifiers
      - Variables
      - Data types
      - Operators etc

Identifiers

- Are names given to anything that you want to identify in a program
- Helps to refer to that item from any place in the program
- Can start with an underscore (_) or a upper or lower case alphabet
- Can have digits
- Identifiers cannot match any of Python's reserved words
- Are case-sensitive

```
bill_id
customer_id
bill_amount
```

Variables

- An identifier for the data and it holds data in your program
- Is a location (or set of locations) in memory where a value can be stored
- A quantity that can change during program execution
- **No declaration of variables**
- **Data type of a variable can change during program execution compared to other strongly typed languages such as Java, C++, C**

```
customer_id = 101                                   # Integer
customer_name = "John"                              # String
bill_amount = 675.45                                # Floating-point
x = 5.3 + 0.9j                                      # complex number
print(customer_id, customer_name, bill_amount)      #prints 101 John 675.45
print(x.real)                                       #prints 5.3
print(x.imag + 3)                                   #prints 3.9
```

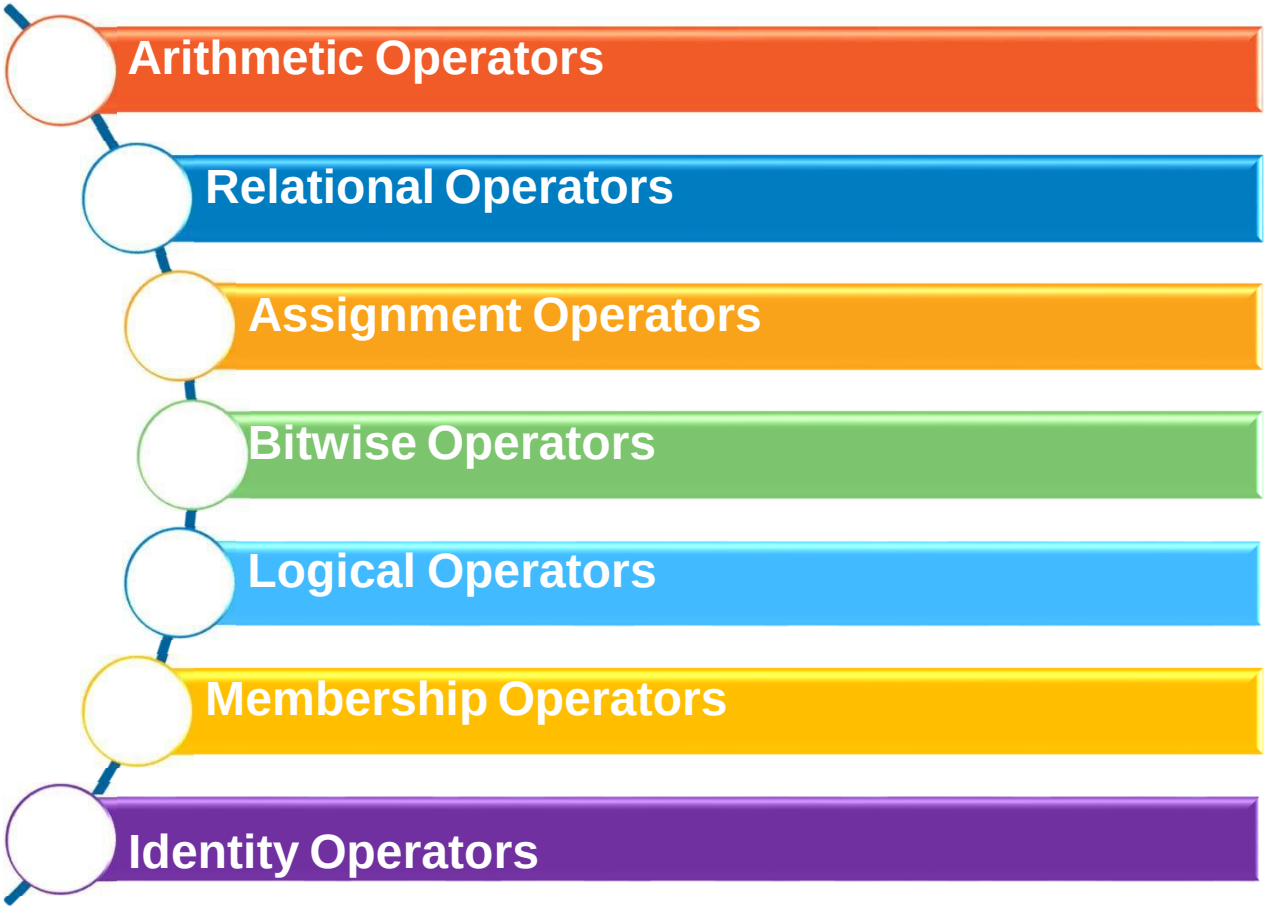
Data Types in Python

Category	Data Type	Example
Integer Type	int	675
	long	9669737712
	complex	2 + 5i
Floating Type	float	642.43
Textual	char	C
	String	Python
Logical	boolean	True, False

Python Operators

Operators (1 of 7)

- Used to perform specific operations on one or more operands (or variables) and provide a result



Operators (2 of 7)

- **Arithmetic Operators**
  - Used for performing arithmetic operations

Operators	Description	Example
+	Additive operator (also used for String concatenation)	2 + 3 = 5
-	Subtraction operator	5 – 3 = 2
*	Multiplication operator	5 * 3 = 15
/	Division operator	6 / 2 = 3
%	Modulus operator	7 % 2 = 1
//	Truncation division (also known as floor division)	10 // 3 = 3 10.0 // 3 = 3.0
**	Exponentiation	10 ** 3 = 1000

Operators (3 of 7)

- **Relational Operators**
  - Also known as **Comparison operators**
  - Used in conditional statements to compare values and take action depending on the result

Operators	Description
==	Equal to
<	Less than
>	Greater than
<=	Lesser than or equal to
>=	Greater than or equal to
!=	Not equal to
<>	Similar to Not equal to

Operators (4 of 7)

- **Assignment Operators**

Operators	Description	Example	Equivalent
=	Assignment from right side operand to left side	c = 50; c = a;	
+=	Add & assigns result to left operand	c += a	c = c + a
-=	Subtract & assigns result to left operand	c -= a	c = c – a
*=	Multiply & assigns result to left operand	c *= a	c = c * a
/=	Divide & assigns result to left operand	c /= a	c = c / a
%=	Calculates remainder & assigns result to left operand	c %= a	c = c % a
//=	Performs floor division & assigns result to left operand	c //= a	c = c // a
**=	Performs exponential calculation & assigns result to left operand	c **= a	c = c ** a

- **Multiple Assignments** – Same value can be assigned to more than one variable

Ex.1: Students Ram, Sham, John belong to semester 6
Ram = Sham = John = 6

Ex.2: a, b, c = 10, 20, 30 is same as a = 10, b = 20, c = 30

Operators (5 of 7)

- **Bitwise Operators**
  - performs bit by bit operation on bits

Operators	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary Ones Complement
<<	Binary Left Shift
>>	Binary Right Shift

Operators (6 of 7)

- **Logical Operators**
  - Are based on Boolean Algebra
  - Returns result as either True or False

Operator	Meaning
and	Short Circuit-AND
or	Short Circuit-OR
not	Unary NOT

Operators (7 of 7)

- **Membership Operators**
  - Checks for membership in a sequence of Strings, Lists, Dictionaries or Tuples

Operators	Description
in	Returns to true if it finds a variable in given sequence else false
not in	Returns to true if it does not find a variable in given sequence else false

- **Identity Operators**
  - Are used to compare memory locations of 2 objects

Operators	Description
is	Returns to true if variables on either side of operator are referring to same object else false
is not	Returns to false if variables on either side of operator are referring to same object else true

Built-in function: id()

- **Id(object)**
  - Returns identity of an object. It is the address of object in memory
  - It will be unique and constant throughout the lifetime of an object

Example:

```
a = 10
b = a
print("Value of a and b before increment")
print("id of a: ",id(a))
print("id of b: ",id(b))
b = a + 1
print("Value of a and b after increment")
print("id of a: ",id(a))
print("id of b: ",id(b))
```

Output

```
Value of a and b before increment
id of a: 1815592664
id of b:  1815592664
Value of a and b after increment
id of a:  1815592664
id of b:  1815592680
```

Note the change in address of variable 'b' after increment

Built-in function: type()

- Used to identify the type of object

Example:

```
int_a = 10

print("Type of 'int_a':", type(int_a))


str_b = "Hello"

print("Type of 'str_b':", type(str_b))


list_c = []

print("Type of 'list_c':", type(list_c))
```

Output:

```
Type of 'int_a': <class 'int'>

Type of 'str_b': <class 'str'>

Type of 'list_c': <class 'list'>
```

Note: Every variable in Python is a object

Coding Standards in Python

- Set of guidelines
  - To Enhance the readability and Clarity of the program
  - Make it easy to debug and maintain the program
- All the letters in a variable name should be in lowercase
- When there are more than two words in variable name, underscore can be used between internal words
- Use meaningful names for variables
- Limit all lines to a maximum of 79 characters
- A function and class should be separated by 2 blank lines
- Methods within classes should be separated by single blank line
- Always surround binary operators with a space on either side:

  Ex: a = a + 1;

Coding Standards in Python

Bad Code	Good Code
a = 10 b = 23 C = 24 sum = a + b + c avg = sum/3 print(“Average: “, avg)	marks1 = 10 marks2 = 23 marks3 = 24 sum_of_marks = marks1 + marks2 + marks3 avg_of_marks = sum_of_marks / 3 print (“Average: “, avg_of_marks)

Day - 2

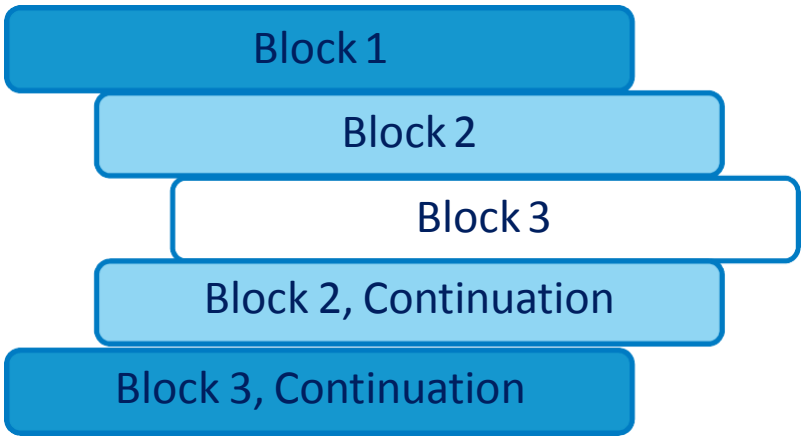
Python Control Structures

Indentation in Python

- Python uses offside rule notation for coding
- Uses indentation for blocks, instead of curly brackets
- The delimiter followed in Python is a colon (:) and indented spaces or tabs.

Example:

```
x = 3
if x > 5:
        print("true")
        print(x)
else:
        print ("false")
print ("Out of if block")
```



1st level indent ⓪ if block
2nd level indent ⓪statements within if or else block

Control Structures

- **Decision making statements**
  - **if statement:**
    - **if** statement checks for a condition and if that is found true a particular set of instructions gets executed

Example:

```
x = 8
if x < 10:
    print("Value of x is %d" %x)
var = 10
if var > 5:
    print    ("Hi")          # line belongs to if block
print("I'm out of if")
```

Output:

```
Value of x is 8
Hi
I'm out of if
```

Syntax:

```
if condition1:
            statement(s)
else:
            statement(s)
```

Predict the output of this code snippet when value of x = 15?

Control Structures…

- elif statement:
  - **elif** statement is used when there is more than one condition to be checked separately

Syntax:

```
if condition1:
        statement(s)
elif condition2:
        statement(s)
else:
        statement(s)
```

Example:

```
var=10
if var > 10 :
    print("Hello")
    print(var)
elif var < 10:
    print("Hola in Spanish for Hello")
    print(var)
else:
    print("Hi")
    print(var)
print("End of Program")
```

Output:

```
Hi
10
End of Program
```

Note: There is no switch case statement in Python unlike C/C++ language

Iterative Statements

- **Loop statements:**
  - Allows us to execute a statement or group of statements multiple times.
    - **While Loop**
    - **For Loop**
    - **Range**

- **Loop Control Statements:**
  - Are used to change flow of execution from its normal sequence.
    - **Break**
    - **Continue**
    - **Pass**

Iterative Statements

– **while loop**:

- Repeats a statement or group of statements while a given condition is TRUE.
- Tests the condition before executing the loop body.

Syntax:

```
while condition:
        statement(s)
```

Example:

```
n = 5

result = 0
counter = 1
while counter <= n:
    result = result + counter
    counter += 1

print("Sum of 1 until %d: %d" % (n, result))
```

Output:

```
Sum of 1 until 5: 15
```

Iterative Statements

– **for loop:**

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Syntax:

```
for iterating_var in sequence:
        statement(s)
```

Example:

```
for counter in 1,2,'Sita', 7,'Ram',5:
    print(counter)
```

Output:

```
1
2
Sita
7
Ram
5
```

Iterative Statements

– range function in loops

- Used in case the need is to iterate over a specific number of times within a given range in steps/intervals mentioned

Syntax: **range(lower limit, upper limit, Increment/decrement by)**

Loop	Output	Remarks
for value in range(1,6): print(value)	1 2 3 4 5	Prints all the values in given range exclusive of upper limit
for value in range(0,6,2): print(value)	0 2 4	Prints values in given range in increments of 2
for value in range(6,1,-2): print(value)	6 4 2	Prints values in given range in decrements of 2
for ch in *"Hello World": print(ch.upper())*	H E L L O W O R L D	Prints all the characters in the string converting them to upper case

Iterative Statements- break

- Loop Control Statements - break and continue
  - When an external condition is triggered, Exits a loop immediately.
  - **Break Statement**:
    - Terminates the loop statement and transfers execution to the statement immediately following the loop.

Example:

```
var = 3
while var > 0:
        print ("I'm in iteration ",var)
        var -= 1
        if var == 2:
            break
        print ("I'm still in while")
print ("I'm out of while loop")
```

Output:

```
I'm in iteration 3  I'm
out of while loop
```

Observe the output of this code snippet when value of var = 5?

Iterative Statements - continue

– **continue** statement:

- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Example:

```
var = 3
while var > 0:
    print ("I'm in iteration ", var)
    var -= 1
    if var == 2:
        continue
        print ("I'm still in if block")
    print ("I'm still in while")
print ("I'm out of while loop")
```

Output:

```
I'm in iteration 3
I'm in iteration 2
I'm still in while
I'm in iteration 1
I'm still in while
I'm out of while loop
```

Iterative Statements- pass

- **pass** statement:
  - **pass** statement is never executed.
  - Used when a statement is required syntactically but do not want any command or code to execute or if the code need to be implemented in future.
  - Behaves like a placeholder for future code

Example:

```
x = "Joy"
if x == "John":
    print ("Name:",x)
elif x == "Joy":
    pass
else:
    print ("in else")
```

Output:

No Output