BCA SEM-VI

Subject: Python Programming (BCA-601)

Created by: Smit Trivedi (Asst.Prof BCA/PGDCA College Idar)

UNIT 1

Beginning with Python

Python is a High-level , Popular Programming Language.

It is a Object-Oriented Scripting Language.

It was Developed by Guido Van Rossum and Released in 1991.

It is Used For : Web Development (Server –Side)

Software Development

Mathematics

System Scripting.

Q- What is the Correct File Extension For Python Files?

1 .pp 2 .pt 3 .py

Ans : .py

Features of Python

1. Free and Open Source: Python language is freely available at the official website and you can download it from the given download link below click on the **Download & Install Python** keyword.
2. Easy to Use (code) : Python has few keywords simple structure and a clearly defined.
3. Easy to Read : Python code is more clearly defined and visible to the eyes.
4. Object-Oriented Language : Structures supports such concepts as polymorphism, operation overloading and multiple inheritance.
5. High- Level language : Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

6. Large Standard Library : Python's bulk of the library is very portable and cross platform compatible on UNIX & WINDOWS.
7. Databases : Python provides interfaces to all major commercial databases.
8. Frontend and backend development :With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like JavaScript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

History of Python

- Python was developed by Guido van Rossum in the 1991 at the National Institute for mathematics and computer science in the Country of Netherlands.
- Python is derived from many other languages including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other Scripting Languages.
- Python is copyrighted. Like Perl, Python source code is now available GPL (General Public License).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.
- Python 1.0 was released in 1994 such as Lambda, Map, Filter and Reduce. And Exception Handling way to handle errors and exception in code.

Early growth and development (1990s):

1991:.....

1994:.....

Establishing a Strong Foundation :

1995-2000 :Python 1.5 has Standard library enhancements and Unicode supports.

Python 2.0 (2000) : Introduced Significant new features like list comprehensions, garbage collection via reference counting, and Unicode support. (means object's references count is zero it is deleted from the memory. But it is part of a circular reference it is moved to generation 1.)

The Shift to Python 3 :

Python 3.0 (2008) **Print Function:** print() became a function, enhancing consistency and flexibility.

New Syntax and Semantics and Removal of Deprecated Features: Simplification of the language by removing outdated features.

Modern era and python's rise (2010s -Present):

2010s Libraries like TensorFlow, Pandas, and Flask emerged, solidifying Python's dominance in machine learning, data analysis, and web development.

End of Python 2: Python 2 reached its end of life on January 1, 2020, marking a complete shift to Python 3.

The latest version of Python is 3.13.0, which was released on October 7, 2024. It includes new features and optimizations compared to Python 3.12.

Python Today : Python remains one of the most popular programming languages globally.

Simplicity: Easy-to-read syntax.

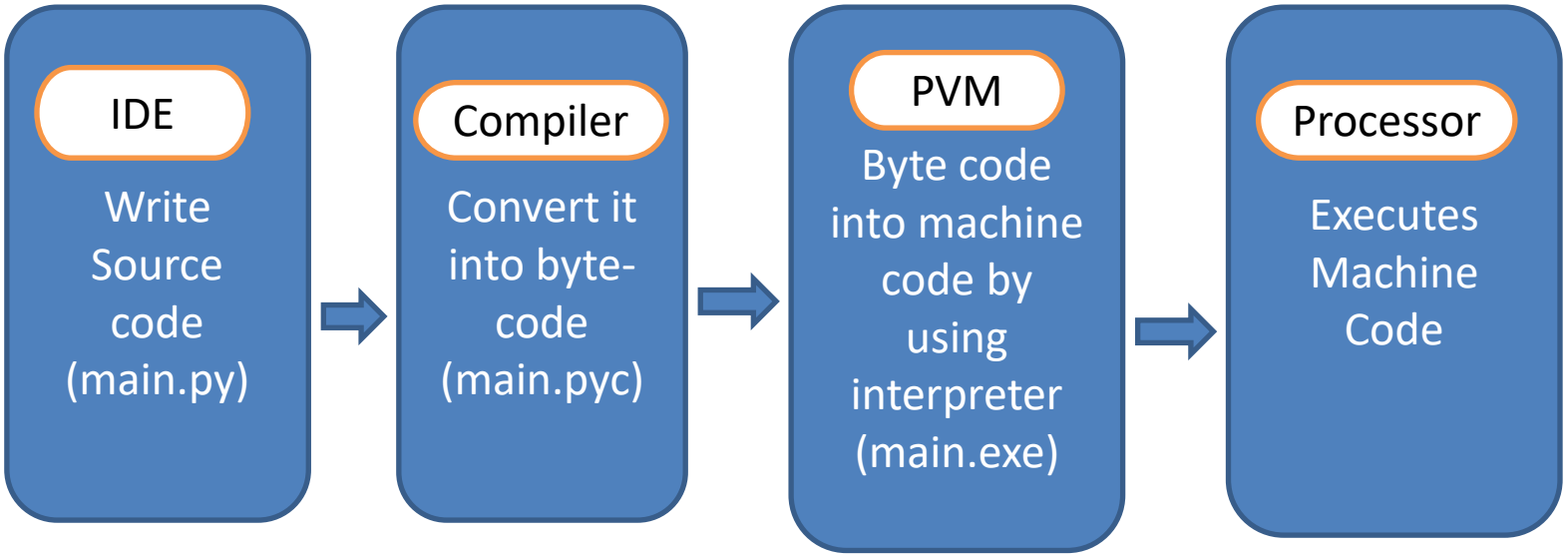
Versatility: Used across diverse domains like web development, AI, scientific computing, and more.

Python's development is now overseen by the **Python Software Foundation (PSF)** with contribution from a global community of developers.

Python Virtual Machine (PVM)

1 It is a Program Which Provides Programming environment.

2 It converts Python byte-code instructions into machine code.



3 so the computer can execute those machine code instructions and display the output.

- So PVM is nothing but a software/interpreter that converts the byte code to machine code for given operating system.

- PVM is also called Python Interpreter and this is the reason Python is called an Interpreted language. We can't see the Byte Code of the program because this happens internally in memory.

Features the PVM:

1. **Portability**:

   Since byte code is platform-independent, Python programs can run on any system with a compatible PVM.

2. **Dynamic Execution**:

   The PVM allows Python's dynamic features, such as runtime modifications to classes or functions.

3. **Error Handling**:

   Errors in the byte code execution are raised as exceptions in the PVM.

4. **Integration with Other Components**:

   The PVM interacts with the Python standard library, user-defined modules, and external libraries.

PVM example:

Below code defines a function add that returns the sum of two numbers. It then calls this function with 3 and 5 as arguments, storing the result in a variable result, and prints it.

```
#  def add(a, b):
        return a + b
    result = add(3, 5)
        print(result)
```

Memory Management in Python

Memory management is very important for software developers to work efficiently with any programming language.

As we know, Python is a famous and widely used programming language. It is used almost in every technical domain. In contrast to a programming language, memory management is related to writing memory-efficient code.

Python Memory Allocation Memory allocation is an essential part of the memory management for a developer. This process basically allots free space in the computer's virtual memory, and there are two types of virtual memory works while executing programs.

(1) Static Memory Allocation

(2) Dynamic Memory Allocation

Type of Memory Allocation

Exact size and type of memory should be known before compile time

memory allocation happens at runtime.

(1) Static Memory : Static memory allocation happens at the compile time. For example - In C/C++, we declare a static array with the fixed sizes. Memory is allocated at the time of compilation. However, we cannot use the memory again in the further program.

ex: **static int A = 10**

Stack Allocation : The Stack data structure is used to store the static memory. It is only needed inside the particular function or method call.

(2) Dynamic Memory allocation:

Unlike static memory allocation, Dynamic memory allocates the memory at the runtime to the program. For example - In C/C++, there is a predefined size of the integer of float data type but there is no predefine size of the data types. Memory is allocated to the objects at the run time. **We use the Heap for implement dynamic memory management.** We can use the memory throughout the program.

ex: int *a

p = new int

- Static memory allocation

```
def my_function ():
 x = 5
y = True
z = 'Hello'
return x, y, z
print(my_function ())
 print(x, y, z)
```

Dynamic memory allocation :

```
a = [0]*10
 print(a)
```

Garbage collection in Python

- Garbage collection in Python is the process of automatically reclaiming memory by cleaning up objects that are no longer in use. Python's garbage collector is part of the **automatic memory management system**, which ensures that memory used by objects that are no longer accessible is released back to the system.
- Python delete unnecessary object automatically to free memory space. This process termed as garbage collection.
- Python garbage collector runs during program execution and it triggered when an objects reference count reaches zero.
- An object reference count increases when it is assigned a new name or placed in container (list ,tuple or dictionary)
- An objects reference count decreases when its deleted with del, its reference is reassigned or its reference goes out of scope.

Comparisons between C-Java-Python

C (C++)	Java	Python
1.Compiled Programming language.	1.Compiled Programming language.	1.Python is an interpreted based programming language.
2.Low-level (closer to hardware).	2.High-level, object-oriented.	2.High-level, interpreted, and easy-to-read.
3.Supports Operator overloading .	3.Does not support Operator Overloading.	3.Supports Operator overloading .
4.Provide both single and multiple inheritance.	4.Provide partial multiple inheritance using interfaces	4.Provide both single and multiple inheritance.
5.Fast, as it compiles directly to machine code.	5.Slower than C, runs on the JVM.	5.Slowest, due to interpretation.
6.Platform dependent .	6.Platform Independent .	6.Platform Independent .
7.Does Not support threads.	7.Has in build multithreading support.	7.Supports multithreading
8.Has limited number of library support	8.Has library support for many concepts like UI.	8.Has a huge set of libraries that make it fit for AI, data science.

Comparisons between C-Java-Python

C (C++)	Java	Python
9.Functions and variables are used outside the class	9.Every bit of code is inside a class.	9.Functions and variables can be declared and used outside the class also.
10.Manual (pointers, malloc, free).	10.Automatic (garbage collection).	10.Automatic (garbage collection).
11.Risk of memory leaks, unsafe pointer usage.	11.Safer, no direct pointer usage.	11.Very safe, managed memory.
12.Fewer standard libraries.	12.Rich standard libraries for enterprise.	12.Extensive standard libraries for various fields.
13.Strictly uses syntax norms Like that ; {}	13.Strictly uses syntax norms Like that , ;	13.Use of ; is not compulsory
14.Operating systems, embedded systems, drivers, low-level programming.	14.Enterprise applications, Android apps, large systems.	14.Data science, web development, automation, AI/ML.

Writing first python program

- How does this work..
- Write a Python Program to Display Hello world

  print (“Hello World”)

  Output Hello World

➡ Comments in Python: Single Line Comments

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Example

```
#This is a comment
print("Hello, World!")
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

- Multi Line Comments: Python does not really have a syntax for multi line comments. To add a multiline comment you could insert a # for each line:
- Example: #This is a comment
- #written in
- #more than just one line
- print("Hello, World!")

Or, not quite as intended, you can use a multiline string. Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example :

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!“)
```

Python variable

- A variable is name attached to a value which can be changed and is later in the code.

- No need of declaration.
- Value of a variable can be changed..
- **Rules for Python Variable Names:**
- Variable names must start with a letter (a-z, A-Z) or an underscore (_).
- They can contain letters, numbers (0-9), and underscores, but no special characters.
- Variable names are **case-sensitive** (myVar and myvar are different).
- They cannot be a reserved keyword (e.g., if, while, def).

- # Assigning values to variables
- name = “Smit" # String
- age = 28 # Integer
- height = 5.6 # Float
- is_student = True # Boolean

- # Accessing variables
- print(name)
- print(age + 5)

- # Output: Smit
- # Output: 33

Operators

(1 Arithmetic operators (2 logical operators ..

Operators are used to perform operations on variables and values.

+	Addition	X+Y
--	Subtraction	X-Y
*	Multiplication	X*Y
/	Division	X/Y
%	Modulus	X%Y
**	Exponentiation	X**Y
//	Floor division	X//Y

Logical Operators: Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Data-types in Python

Built-in data types : Built-in Data types are those data types that are pre-defined by the programming language. Most languages have native data types that are created for easy of execution of data. Such Data types are called Built-in Data Types. These data types can be used directly in the program without the hassle of creating them.

Text Type:	str
Numeric Types:	**int , float, complex**
Sequence Types:	**list, tuple, range**
Mapping Type:	**dict**
Set Types:	**set, frozen set**
Boolean Type:	**bool**
Binary Types:	**bytes, bytearray, memoryview**
None Type:	**None Type**

- Strings : 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. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.
- For ex : str = ‘Hello World!’
- print str # Prints complete string
- print str[0] # Prints first character of the string
- print str[2:5] # Prints characters starting from 3rd to 5th
- print str[2:] # Prints string starting from 3rd character
- print str * 2 # Prints string two times
- print str + "TEST" # Prints concatenated string

- This will produce the following results.

Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST

String methods

Method	Description
Capitalize()	Converts the first character to upper case
Casefold()	Converts string into lower case
Center()	Returns a centered string
Count()	Returns the number of times a specified value occurs in a string
Encode()	Returns an encoded version of the string
Endswitch()	Returns true if the string ends with the specified value
Expandtabs()	Sets the tab size of the string

Find()	**Searches the string for a specified value and returns the position of where it was found**
Format()	Formats specified values in a string
Format_map()	Formats specified values in a string
Index()	Searches the string for a specified value and returns the position of where it was found
Isalnum()	Returns True if all characters in the string are alphanumeric
Isalpha()	**Returns True if all characters in the string are in the alphabet**
Isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
Isdigit()	Returns True if all characters in the string are digits
Isidentifier()	Returns True if the string is an identifier
Islower()	Returns True if all characters in the string are lower case
Isnumeric()	Returns True if all characters in the string are numeric
Isprintable()	Returns True if all characters in the string are printable

Isspace()	Returns True if all characters in the string are whitespaces
Istitle()	Returns True if the string follows the rules of a title
Isupper()	Returns True if all characters in the string are upper case
Join()	Converts the elements of an iterable into a string
Lower()	Converts a string into lower case
Partition()	Returns a tuple where the string is parted into three parts
Replace()	Returns a string where a specified value is replaced with a specified value
Zfill()	Fills the string with a specified number of 0 values at the beginning

Numeric types

Python includes three numeric types to represent numbers: integers, float, and complex number.

Number data types store numeric values. Number objects are created when you assign a value to them.

example : int integer numbers

x = 42 # Example of an int

example: float Floating-point numbers(decimal numbers).

y= 3.14 # Example of a float

example : Complex numbers with real and imaginary parts.

z = 2 + 3j # Example of a complex number

Int	Long	Float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Explicit conversion of datatypes

- Explicit data type conversion, also known as type casting, is a technique in Python that allows a programmer to manually change a value from one data type to another.
- This is done by using specific functions or operators to explicitly specify the target data type in the code.
- Here are some examples of explicit data type conversion functions in Python:
- **int()**: Converts a data type to an integer, removing any fractional part. For example, int(5.9) produces 5
- **float()**: Converts a data type to a float. For example, float(2) produces 2.0
- **str()**: Converts a data type to a string. For example, str(3.14) produces "3.14"

Function	Description
int(x [,base]) tuple(s)	Converts x to an integer. base specifies the base if x is a string Converts to a tuple.
long(x [,base]) dict(d)	Converts x to a long integer. base specifies the base if x is a string. Creates a dictionary. d must be a sequence of (key,value) tuples.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number
str(x) Hex(x)	Converts object x to a string representation Converts an integer to a hexadecimal string
repr(x)	Converts object x to an expression string
eval(str) oct(x)	Evaluates a string and returns an object Converts an integer to an octal string.

You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1
y = 2.8
z = 1j
a = float (x)          #class float
b = int (y)            #class int
c = complex(x)         #variable
print(a)               #variable a float
print(b)               # variable b int
print(c)               # variable c complex
print(type(a))
print(type(b))
print(type(c))
```

Sequences Types:

List	Tuple	Range	set datatype	Frozenset	Mapping types	Boolean types	Binary types
Float	Count	Bytes	Concatenation	Dictionary	Indexing	Slicing	Length

A Sequence is an ordered Collection of items in Python Language and each item has index label of integer type.
In Python three Sequences data type is available like : String , List , Tuple…

String : String is a Sequence of character or group of character & character in a String has indexed with integer.
For Example : Sname = “Computer”

index : 0 1 2 3 4 5 6 7
value: C o m p u t e r

- **List** : List is a Sequence of items separated by commas and items are enclosed in square braces [ ]. In list items may be different data type & it is mutable..
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.
- for eg:list1 [44, 12.6, “Banglore”,”N115”]

Tuple: Tuple is a Sequence of items separated by commas and items are enclosed in round braces (). In list items may be different data type & it is immutable..

Tuples are used to store multiple items in a single variable.

eg: mytuple = ("apple", "banana", "cherry")

eg: tpl = (1,2,3, “a”, “b”)

print (tpl)

Range : The Range Data type is used to represent a Sequence of number. Using a for loop with range() we can repeat an action a specific number of times.

let's see how to use the range() function of Python 3 to produce the first six numbers.

Syntax : range(start, stop, step)

eg: r = range (1 , 10, 2) # start=1, stop=10, step=2

print(list(r))

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop
step	Optional. An integer number specifying the incrementation. Default is 1

For ex 1: x = range (6)

```
for n in x:
print(n)
```

For ex 2: x = range (3 , 20, 2)

```
for n in x:
print(n)
```

Set Datatype

Set: Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

- A set is a collection which is unordered, unchangeable*, and unindexed.
- Sets are written with curly brackets.

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

- There are four main concepts in Set Datatype.

- Set Items :Set items are unordered, unchangeable, and do not allow duplicate values.
- Unordered: Unordered means that the items in a set do not have a defined order. Set items can appear in a different order every time you use them, and cannot be referred to by index or key.
- Unchangeable Set items are unchangeable, meaning that we cannot change the items after the set has been created. Once a set is created, you cannot change its items, but you can remove items and add new items
- Duplicates Not Allowed : Sets cannot have two items with the same value.
- Example: Duplicate values will be ignored:
- **thisset = {"apple", "banana", "cherry", "apple"}**
- **print(thisset)**

Mapping types

- Mapping in Python is an unordered data type.
- Dictionary is the only mapping data type available in python.
- Dictionary : in Dictionary data items are stored In pairs. Value and items are enclosed in curly braces{ }. Every key is separated from its value with colon (:) & each item is separated by comma.
- Foreg:
  dict={'JAN':31,'FEB':28,'MAR':31,'APR':30}
  print(dict)

Frozen set

- The frozenset() is an inbuilt function in Python which takes an iterable object as input and makes them immutable. Simply it freezes the iterable objects and makes them unchangeable.
- This function takes input as any iterable object and converts them into an immutable object.
- The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable). The frozen sets are the immutable form of the normal sets. It means we cannot remove or add any item into the frozen set.
- Syntax :**frozenset(iterable)**

Parameter	Description
iterable	An iterable object, like list, set, tuple etc.

Boolean types

- The Boolean data type is either True or False. In Python, Boolean variables are defined by the True and False keywords. The output <class 'bool'> indicates the variable is a Boolean data type. Note the keywords True and False must have an Upper Case first letter.
- For example, 1== 0 is **True** whereas 2<1 is **False.**
- x=True
- y=False
- print(x)
- print(y)
- print(bool(1))
- print(bool(0))

Binary types

- A data type defines the type of a variable. Since everything is an object in Python, data types are actually classes; and the variables are instances of the classes.

  bytes, bytearray, memoryview:

  bytes and **bytearray** are used for manipulating binary data. The memoryview uses the buffer protocol to access the memory of other binary objects without needing to make a copy.

  Bytes objects are immutable sequences of single bytes. We should use them only when working with **ASCII** compatible data.

Ex: bytearray:

- x = bytearray(5)
- print(x)
- print(type(x))

Ex: byte

```
x = b"Hello“
print(x)
print(type(x))
x = memoryview(bytes(5))
print(x)
print(type(x))
```

```
X  = 5
Y = 10
X = input(‘enter value of x: ‘)
Y = input(‘enter value of y: ‘)
Temp = x
X = y
Y = temp
Print(‘ the value of x after swapping : {}’.format(x) )
Print(‘ the value of y after swapping : {}’.format(y) )
```