

Python

RLMCA369 Python

Programming

MCA – Semester -5

Part-1

UNIT-1

- **Introduction to Python:** Features of Python, How to Run Python, Identifiers, Reserved Keywords, Variables, Input, Output and Import Functions, Operators
- **Data Types:** Numbers, Strings, List, Tuple, Set, Dictionary, Data Type Conversions.
- **Decision Making,** Loops, Nested Loops, Control Statements, Types of Loops

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by Guido Van Rossum at CWI in Netherland.
- In 1994, Python 1.0 was released
- On December 3, 2008, Python 3.0 (also called "Py3K") was released.
- Python 3.8.5, documentation released on 20 July 2020

- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages: *ABC language and Modula-3*

Lets Start- Python

- **Python** is a general purpose, dynamic, high level, and interpreted programming language.
- It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.

Easy to Learn and Use

- Python is easy to learn and use. It is developer-friendly and high level programming language.

Expressive Language

- Python language is more expressive means that it is more understandable and readable.

Interpreted Language

- Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

Cross-platform Language

- Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

Free and Open Source

- Python language is freely available at [official web address](#). The source-code is also available. Therefore it is open source.

Object-Oriented Language

- Python supports object oriented language and concepts of classes and objects come into existence.

Extensible

- It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

Large Standard Library

- Python has a large and broad library and provides rich set of module and functions for rapid application development.

GUI Programming Support

- Graphical user interfaces can be developed using Python.

Integrated

- It can be easily integrated with languages like C, C++, JAVA etc.

Python Applications

Web Applications

- We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, BeautifulSoup, Feedparser etc.
- It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

Desktop GUI Applications

- Python provides GUI library to develop user interface in python based application.
- Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms.
- The **Kivy** is popular for writing multitouch applications.

Software Development

- Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

Scientific and Numeric

- Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

Business Applications

- Python is used to build Business applications like ERP and e-commerce systems. Tryton is a high level application platform.

Console Based Application

- We can use Python to develop console based applications. For example: **IPython**.

Audio or Video based Applications

- Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

3D CAD Applications

- To create CAD application Fandango is a real application which provides full features of CAD.

Enterprise Applications

- Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

Applications for Images

- Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.
- There are several such applications which can be developed using Python

- **The language is used by companies in real revenue generating products, such as:**
- ☆ In operations of Google search engine, youtube, etc.
- ☆ Bit Torrent peer to peer file sharing is written using Python
- ☆ Intel, Cisco, HP, IBM, etc use Python for hardware testing.
- ☆ Maya provides a Python scripting API
- ☆ i-Robot uses Python to develop commercial Robot.
- ☆ NASA and others use Python for their scientific programming task.

Python Programming

- To write and run Python program, we need to have Python interpreter installed in our computer.
- **IDLE (GUI integrated)** is the standard, most popular Python development environment.
- **IDLE** is an acronym of Integrated Development Environment.
- It lets edit, run, browse and debug Python Programs from a single interface. This environment makes it easy to write programs

- Python shell can be used in two ways, viz., **interactive mode and script mode**.
- Where **Interactive Mode**, as the name suggests, allows us to interact with OS;
- **Script mode** let us create and edit python source file.

- **Interactive Mode**
- What we see is a welcome message of Python interpreter with revision details and the Python prompt, i.e >>>
- This is a primary prompt indicating that the interpreter is expecting a python command

- We may try the following and check the response:
- i) `print(5+7)`
- (ii) `5+7`
- iii) `6*250/9`
- iv) `print (5-7)`

- It is also possible to get a sequence of instructions executed through interpreter
- **Example 1**
 - `>>> x=2`
 - `>>> y=6`
 - `>>> z = x+y`
 - `>>> print (z)`
 - **8**

- **Example 2**

- `>>> a=3`

- `>>> a+1, a-1`

- **`(4,2)` #result is tuple of 2 values**

Script mode

- In script mode, we type Python program in a file and then use the interpreter to execute the content from the file.
- Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately.
- But for coding more than few lines, we should always save our code so that we may modify and reuse the code.

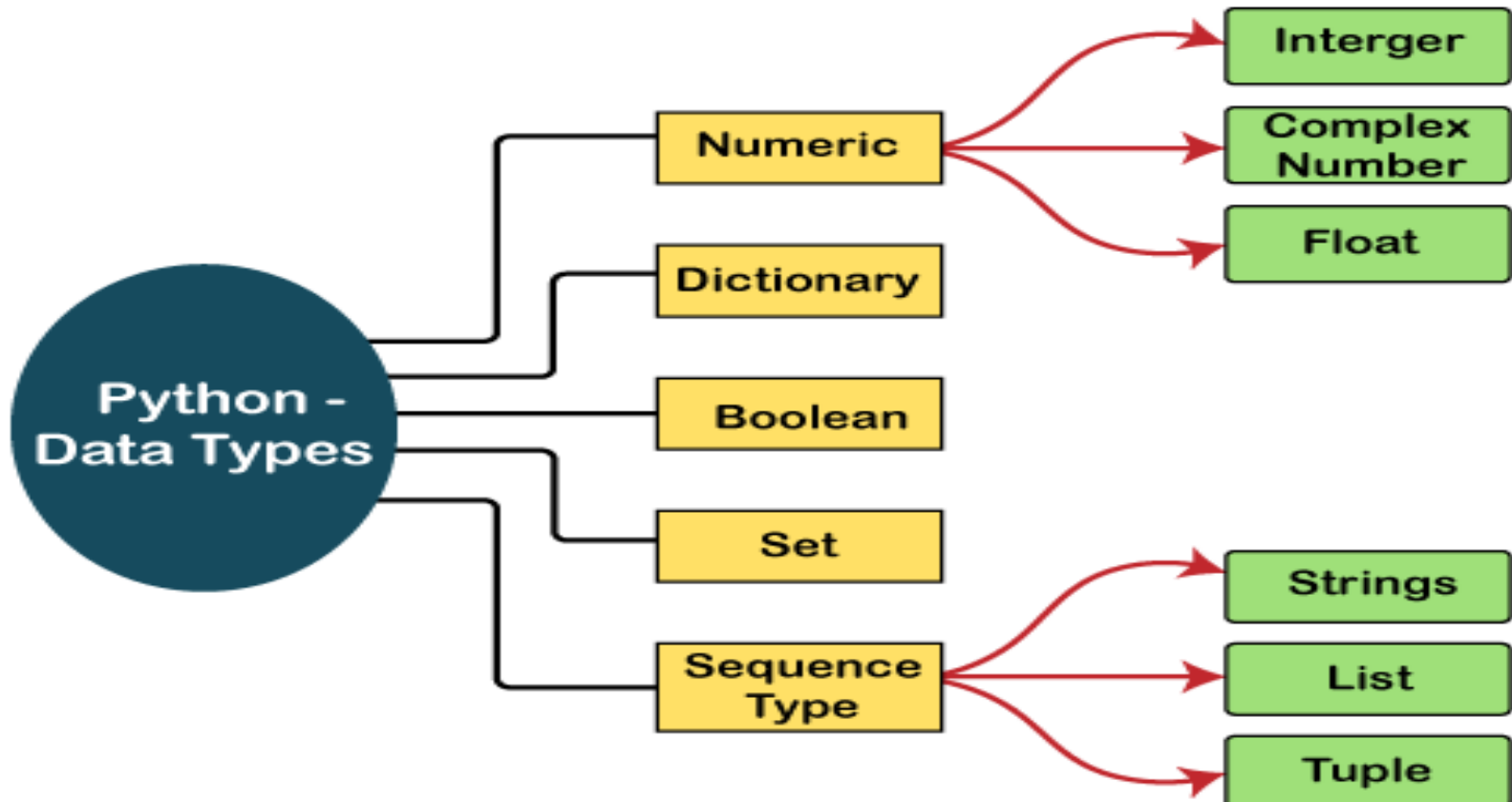
Variables and DataTypes

- **Identity of the object:** It is the object's address in memory and does not change once it has been created.
- *(We would be referring to objects as variable for now)*
- Variables can hold values of different data types. **Python is a dynamically typed language** hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Identity of the object: It is the object's address in memory and does not change once it has been created.

(We would be referring to objects as variable for now)

Type (i.e data type): It is a set of values, and the allowable operations on those values. It can be one of the following:



1. Number

- Number data type stores Numerical Values. This data type is immutable i.e. value of its object cannot be changed (we will talk about this aspect later). These are of three different types:
 - a) Integer & Long
 - b) Float/floating point
 - c) Complex

- `>>> a = 10`
- `>>> type(a)`
`<type 'int'>`
- Integers contain **Boolean Type** which is a unique data type, consisting of two constants, **True & False**. A Boolean True value is **Non-Zero, Non-Null and Non-empty**.

- **Example**
- `>>> flag = True`
- `>>> type(flag)`
- `<type 'bool'>`
- **Floating Point:** Numbers with fractions or decimal point are called floating point numbers.
- -2.0×10^5 will be represented as `-2.0e5`
- 2.0×10^{-5} will be `2.0E-5`

- **Complex:** Complex number in python is made up of two floating point values, one each for real and imaginary part.
- For accessing different parts of variable (object) x; we will use x.real and x.imag. Imaginary part of the number is represented by 'j' instead of 'i', so 1+0j denotes zero imaginary part

- **Example**

```
>>> x = 1+0j
```

```
>>> print (x.real,x.imag)
```

```
1.0 0.0
```

Example

```
>>> y = 9-5j
```

```
>>> print (y.real, y.imag)
```

```
9.0 -5.0
```

- **Sequence**
- A sequence is an ordered collection of items, indexed by positive integers. It is combination of mutable and non mutable data types.
- Three types of sequence data type available in Python are
- **Strings, Lists & Tuples**

- **String:** is an ordered sequence of letters/characters. They are enclosed in single quotes or double.
- The quotes are not part of string. They only tell the computer where the string constant begins and ends.

They can have any character or sign, including space in them. These are immutable data types.

- We will learn about immutable data types while dealing with third aspect of object i.e. value of object.

- `>>> a = 'Ram'`
- A string with length 1 represents a character in Python.
- Conversion from one type to another
- If we are not sure, what is the data type of a value, Python interpreter can tell us:

```
>>> type('Good Morning')
```

```
<type 'str'>
```

```
>>> type('3.2')
```

```
<type 'str'>
```

- It is possible to change one type of value/variable to another type. It is known as **type conversion or type casting**.
- The conversion can be done **explicitly** (programmer specifies the conversions) or **implicitly** (Interpreter automatically converts the data type).
- For **explicit type casting**, we use functions (constructors):
 - `int ()`
 - `float ()`
 - `str ()`
 - `bool ()`

- **Example**

```
>>> a= 12.34
```

```
>>> b= int(a)
```

```
>>> print b
```

```
12
```

- **Example**

```
>>>a=25
```

```
>>>y=float(a)
```

```
>>>print y
```

```
25.0
```

List

- **Lists** are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within **square brackets []**.
- Lists are **mutable i.e., modifiable**.
- We can **use slice [:] operators** to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

```
l = [1, "hi", "python", 2]
```

```
print (l[3:])
```

```
print (l[0:2])
```

```
print (l)
```

```
print (l + l)
```

```
print (l * 3)
```

- **Output:**

```
[2]
```

```
[1, 'hi']
```

```
[1, 'hi', 'python', 2]
```

```
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

```
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Tuple

- A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types.
- The items of the tuple are separated with a comma (,) and **enclosed in parentheses ()**.
- A tuple is a **read-only data structure** as we can't modify the size and value of the items of a tuple.**(Immutable)**

```
t = ("hi", "python", 2)
print (t[1:]);
print (t[0:1]);
print (t);
print (t + t);
print (t * 3);
print (type(t))
t[2] = "hi"; # immutable
```

- **Output:**

('python', 2)

('hi',)

('hi', 'python', 2)

('hi', 'python', 2, 'hi', 'python', 2)

('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)

<type 'tuple'>

Traceback (most recent call last):

File "main.py", line 8, in <module>

t[2] = "hi";

TypeError: 'tuple' object does not support item assignment

Set

- Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }.
- A set is similar to list, except that it cannot have duplicate entries.
- Once created, elements of a set cannot be changed.

#create a set

```
>>> set1 = {10,20,3.14,"New Delhi"}
```

```
>>> print(type(set1))
```

```
<class 'set'>
```

```
>>> print(set1)
```

```
{10, 20, 3.14, "New Delhi"}
```

#duplicate elements are not included in set

```
>>> set2 = {1,2,1,3}
```

```
>>> print(set2)
```

```
{1, 2, 3}
```

None

None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither False nor 0 (zero).

```
>>> myVar = None
```

```
>>> print(type(myVar))
```

```
<class 'NoneType'>
```

```
>>> print(myVar)
```

```
None
```

Mapping

- Mapping is an unordered data type in Python.
- Currently, there is only one standard mapping data type in Python called dictionary.

Dictionary

- Dictionary is an ordered set of a key-value pair of items.
- It is like an associative array or a hash table where each key stores a specific value.
- Key can hold any primitive data type whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma and **enclosed in the curly braces {}.**

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
print("1st name is "+d[1]);
print("2nd name is "+ d[2]);
print (d);
print (d.keys());
print (d.values())
```

- **Output:**

1st name is Jimmy

2nd name is Alex

{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}

[1, 2, 3, 4]

['Jimmy', 'Alex', 'john', 'mike']

Python Keywords

- Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter.
- Each keyword have a special meaning and a specific operation.
- These keywords can't be used as variable.

- Following is the List of Python Keywords.
- True False None and as
- asset def class continue break
- else finally elif del except
- global for if from import
- raise try or return pass
- nonlocal in not is lambda

Python Operators

Python Operators

- The operator can be defined as a symbol which is responsible for a particular operation between two operands.
- Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.
- **Arithmetic operators**
- **Comparison operators**
- **Assignment Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Membership Operators**
- **Identity Operators**

Arithmetic operators

- Arithmetic operators are used to perform arithmetic operations between two operands.
- It includes +(addition), -(subtraction), *(multiplication), /(divide), %(remainder), //(floor division), and exponent (**).
- **+ (Addition)**
- It is used to add two operands. For example, if $a = 20$, $b = 10$ $a+b = 30$

- (Subtraction)

- It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if $a = 20$, $b = 10$
- $a - b = 10$

- **/ (divide)**
- It returns the quotient after dividing the first operand by the second operand. For example, if $a = 20$, $b = 10$
- $a/b = 2$
- *** (Multiplication)**
- It is used to multiply one operand with the other. For example, if $a = 20$, $b = 10$
- $a * b = 200$

- **** (Exponent)**

- It is an exponent operator represented as it calculates the first operand power to second operand.
- $a^{**}3$

- **// (Floor division)**

- It gives the floor value of the quotient produced by dividing the two operands.
- rounds the result down to the nearest whole number
- $x=15$ $y=2$
- $x//y$
- Result is 7

Comparison operators

- Comparison operators are used to comparing the value of the two operands and returns Boolean value true or false accordingly.
- **==** If the value of two operands is equal, then the condition becomes true.
- **!=** If the value of two operands is not equal then the condition becomes true.
- **>=** If the first operand is greater than or equal to the second operand, then the condition becomes true.

- \neq If the value of two operands are not equal, then the condition becomes true.
- $>$ If the first operand is greater than the second operand, then the condition becomes true.
- $<$ If the first operand is less than the second operand, then the condition becomes true.

Python assignment operators

Python assignment operators

- The assignment operators are used to assign the value of the right expression to the left operand.
- **=** It assigns the value of the right expression to the left operand.
- **+=** It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand.
For example, if $a = 10$, $b = 20 \Rightarrow a + = b$ will be equal to $a = a + b$ and therefore, $a = 30$.
- **-=** It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand.
For example, if $a = 20$, $b = 10 \Rightarrow a - = b$ will be equal to $a = a - b$ and therefore, $a = 10$.
- ***=** It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand.
For example, if $a = 10$, $b = 20 \Rightarrow a * = b$ will be equal to $a = a * b$ and therefore, $a = 200$.

- **%=** It divides the value of the left operand by the value of the right operand and assign the remainder back to left operand. **For example, if $a = 20$, $b = 10 \Rightarrow a \% = b$ will be equal to $a = a \% b$ and therefore, $a = 0$.**
- ****=** $a**=b$ will be equal to $a=a**b$, for example, if $a = 4$, $b = 2$, $a**=b$ will assign $4**2 = 16$ to a .
- **//=** $a//=b$ will be equal to $a = a// b$, for example, if $a = 4$, $b = 3$, $a//=b$ will assign $4//3 = 1$ to a .

Logical Operators

- The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.
- **and** If both the expression are true, then the condition will be true. If a and b are the two expressions, **eg :a → true, b → true => a and b → true.**

- **or** If one of the expressions is true, then the condition will be true. If a and b are the two expressions, eg $a \rightarrow \text{true}$, $b \rightarrow \text{false} \Rightarrow a \text{ or } b \rightarrow \text{true}$.
- **not** If an expression a is true then not (a) will be false and vice versa.

Membership Operators

- Python membership operators are used to check the **membership of value inside a data structure**.
- If the value is present in the data structure, then the resulting value is true otherwise it returns false.
- **in** It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
- **not in** It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Membership Operators

Examples

Result

`in`

`10 in (10, 20, 30)`

`True`

`red in ('red', 'green', 'blue')`

`True`

`not in`

`10 not in (10, 20, 30)`

`False`

Example

```
>>> 'Dr.' in 'Dr. Madison'
```

```
True
```

Python Comments

- Comments in Python can be used to explain any program code. It can also be used to hide the code as well.
- In python, any statement written along with # symbol is known as a comment.
- The interpreter does not interpret the comment.

1) Single Line Comment:

- In case user wants to specify a single line comment, then comment must start with **#**
- **Eg:**

```
# This is single line comment.
```

```
print "Hello Python"
```

2) Multi Line Comment:

- Multi lined comment can be given inside triple quotes.
- **eg:**

```
""" This
```

```
Is
```

```
Multipline comment"""
```

Variable Assignment and Keyboard Input

```
>>> name =input('What is your first name?')
```

What is your first name? John

- In this case, the variable **name** is assigned the string **'John'**.
- If the user hit return without entering any value, name would be assigned to the **empty string ('')**.
- **Note:**
- All input is returned by the input function as a **string type**.
- *For the input of numeric values, the response must be converted to the appropriate type. Python provides built-in type conversion*
- functions **int ()** and **float ()** for this purpose,

Example

```
>>> line =input('How many credits do you have?')
>>>num_credits =int(line)
>>>line= input('What is your grade point average?')
>>>gpa =float(line)
```

Example

- Note that the program lines above could be combined as follows,

```
>>>num_credits = int(input('How many credits do you have? '))
>>>gpa= float(input('What is your grade point average? '))
```

Note

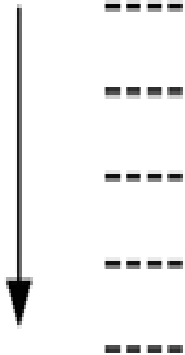
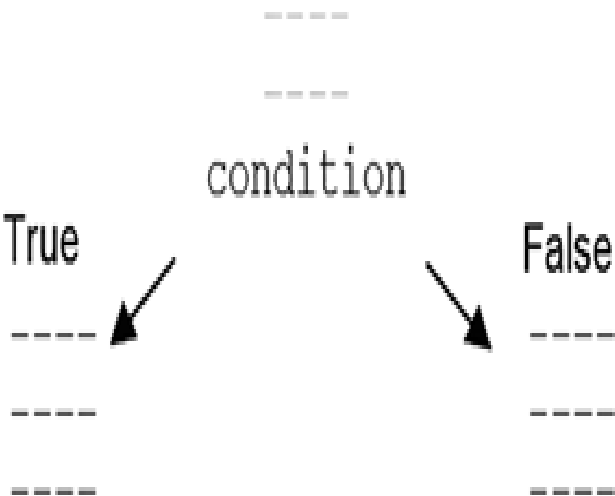
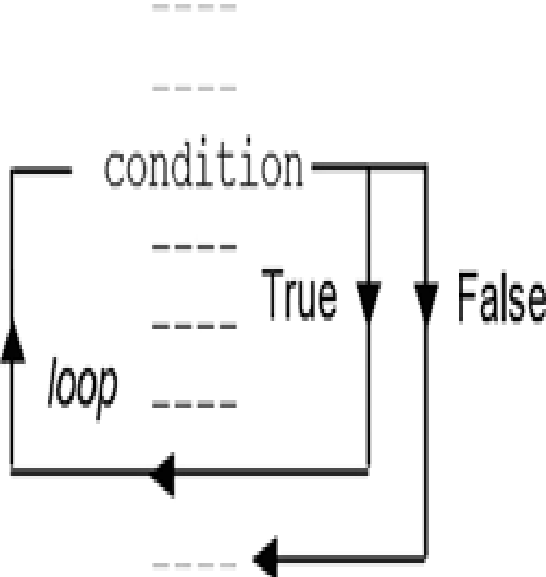
- All input is returned by the input function as a string type.
- Built-in functions **int()** and **float()** can be used to convert a string to a numeric type

Control Structures

- *Control flow is the order that instructions are executed in a program.*
- A ***control statement is a statement*** that determines the control flow of a set of instructions.
- There are three fundamental forms of control that programming languages provide—***sequential control , selection control , and iterative control***

- **Sequential control** is an implicit form of control in which instructions are executed in the order that they are written.
- A program consisting of only sequential control is referred to as a “straight-line program.”
- **Selection control** is provided by a control statement that *selectively executes instructions*,
- **Iterative control** is provided by an iterative control statement that *repeatedly executes instructions*

- Each is based on a given condition.
- Collectively a set of instructions and the control statements controlling their execution is called a **control structure** .

Sequential Control	Selection Control	Iterative Control
		

Selection Control

- A selection control statement is a control statement providing selective execution of instructions.
- *A selection control structure is a given set of instructions and the selection control statement(s) controlling their execution.*
- *We look at the if statement providing selection control in Python*

if statement

Example use

```
if condition:  
    statements  
else:  
    statements
```

```
if grade >= 70:  
    print('passing grade')  
else:  
    print('failing grade')
```

```
if grade == 100:  
    print('perfect score!')
```

Indentation in Python

- In Python, however, indentation is used to associate and group statements
- A header in Python is a specific keyword followed by a **colon(:)**. In the figure, the if-else statement contains two headers

Valid indentation		Invalid indentation	
(a)	<pre> if condition: statement statement else: statement statement </pre>	(b)	<pre> if condition: statement statement else: statement statement </pre>
(c)	<pre> if condition: statement statement else: statement statement </pre>	(d)	<pre> if condition: statement statement else: statement statement </pre>

- A **header** in Python starts with a keyword and ends with a **colon**. The group of statements following a header is called a **suite**. A header and its associated suite are together referred to as a **clause**.

- **Multi-Way Selection**
- In this section, we look at the two means of constructing multi-way selection in Python—one involving multiple nested if statements, and the other involving a **single if statement** and the use of **elif headers**

- **Nested if Statements**

Nested if statements

Example use

```
if condition:
    statements
else:
    if condition:
        statements
    else:
        if condition:
            statements

        etc.

if grade >= 90:
    print('Grade of A')
else:
    if grade >= 80:
        print('Grade of B')
    else:
        if grade >= 70:
            print('Grade of C')
        else:
            if grade >= 60:
                print('Grade of D')
            else:
                print('Grade of F')
```

- **The elif Header in Python**

- If statements may contain only one else header. Thus, if-else statements must be nested to achieve multi-way selection.
- Python, however, has another header called **elif (“else-if”)** that provides multi-way selection in a *single if statement*,

```
if grade >= 90:  
    print('Grade of A')  
elif grade >= 80:  
    print('Grade of B')  
elif grade >= 70:  
    print('Grade of C')  
elif grade >= 60:  
    print('Grade of D')  
else:  
    print('Grade of F')
```

- All the headers of an if-elif statement are indented the same amount, thus avoiding the deeply nested levels of indentation with the use of if-else statements.

A final else clause may be used for “catch-all” situations.

Short Hand If

- One line if statement

Eg:- `if a > b: print("a is greater than b")`

One line if else statement:

Eg:- `print("A") if a > b else print("B")`

One line if else statement, with 3 conditions:

`print("A") if a > b else print("=") if a == b else print("B")`

Iterative Control

- An **iterative control statement** is a control statement providing the repeated execution of a set of instructions.
- An *iterative control structure* is a set of instructions and the iterative control statement(s) controlling their execution.
- Because of their repeated execution, iterative control structures are commonly referred to as **“loops.”**

While Statement

- A while statement is an iterative control statement that repeatedly executes a set of statements
- based on a provided Boolean expression (condition). All iterative control needed in a program can be achieved by use of the **while statement**

while statement	Example use
<pre>while <i>condition</i>: <i>suite</i></pre>	<pre>sum = 0 current = 1 n = int(input('Enter value: ')) while current <= n: sum = sum + current current = current + 1</pre>

an example of a while loop in
Python that sums the first n integers, for a given
(positive) value n entered by the user

while Loop

- With the while loop we can execute a set of statements as **long as a condition is true.**
- **Example to Print i as long as i is less than 6:**

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    i += 1
```

The break Statement

- With the break statement we can stop the loop even if the while condition is true
- **Example** Exit the loop when i is 3:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

- **The continue Statement**
- With the continue statement we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Note that number 3 is missing in the result

Python For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- **Example for Print each fruit in a fruit list:**

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

Example 2

```
for x in "banana":  
    print(x)
```

- **The break Statement**
- With the break statement we can stop the loop before it has looped through all the items:

Break example

- ```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
 print(x)
 if x == "banana":
 break
```
- **Example2- Program to Exit the loop when x is "banana"**

**but this time the break comes before the print**

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
 if x == "banana":
 break
 print(x)
```

Note: Print only apple



- **The continue Statement**

- With the continue statement we can stop the current iteration of the loop, and continue with the next:

- **Example for -Do not print banana**

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
 if x == "banana":
 continue
 print(x)
```

# The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- **Example**
- ```
for x in range(6):  
    print(x)
```
- **Note that range(6) is not the values of 0 to 6, but the values 0 to 5.**

- Example Using the **start** parameter:
- for x in range(2, 6):
 print(x)

The **range()** function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

- Example **Increment the sequence with 3**
(default is 1):

```
for x in range(2, 30, 3):  
    print(x)
```

- **Else in For Loop**
- The else keyword in a for loop specifies a block of code to be executed when the loop is finished:
- **Example -Print all numbers from 0 to 5, and print a message when the loop has ended:**

```
for x in range(6):  
    print(x)  
else:  
    print("Finally finished!")
```

Nested for Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":
- **Example –Print the adjective of every fruit:**

```
adj = ["red", "big", "tasty"]
```

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in adj:
```

```
    for y in fruits:
```

```
        print(x, y)
```

- Output
- red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry

Python break statement

- The break is a keyword in python which is used to bring the program control out of the loop.
- The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.
- In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.
- The break is commonly used in the cases where we need to break the loop for a given condition.

Example

```
list =[1,2,3,4]
count = 1;
for i in list:
    if i == 4:
        print("item matched")
        count = count + 1;
        break
print("found at",count,"location");
```

Python continue Statement

- The continue statement in python is used to bring the program control to the beginning of the loop.
- The continue statement skips the remaining lines of code inside the loop and start with the next iteration.
- It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

Example

```
i=1; #initializing a local variable  
#starting a loop from 1 to 10  
for i in range(1,11):  
    if i==5:  
        continue;  
    print("%d"%i);
```

Python Pass

- In Python, **pass keyword is used to execute nothing**; it means, when we don't want to execute code, the pass can be used to execute empty.
- It is same as the name refers to. It just makes the control to pass by without executing any code.
- If we want to bypass any code pass statement can be used.

Python Pass Example

```
for i in [1,2,3,4,5]:  
    if i==3:  
        pass  
        print "Pass when value is",i  
    print i,
```

- **Output:**

>>>

1 2 Pass when value **is** 3

3 4 5

>>>

Python String

- Most popular data type in python i.e., string.
- In python, strings can be created by enclosing the character or the sequence of characters in the quotes.
- Python allows us to use single quotes, double quotes, or triple quotes to create the string.
- Consider the following example in python to create a string.
- `str = "Hi Python !"`
- Here, if we check the type of the variable str using a python script
- `print(type(str))`, then it will **print** string (str).

- Strings indexing and splitting
- Like other languages, the indexing of the python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

- As shown in python, the slice operator `[]` is used to access the individual characters of the string.
- However, we can use the `:` (colon) operator in python to access the substring. Consider the following example.
-

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[:] = 'HELLO'

str[1] = 'E'

str[0:] = 'HELLO'

str[2] = 'L'

str[:5] = 'HELLO'

str[3] = 'L'

str[:3] = 'HEL'

str[4] = 'O'

str[0:2] = 'HE'

str[1:4] = 'ELL'

- **Reassigning strings**

- Updating the content of the strings is as easy as assigning it to a new string.
- The string object doesn't support item assignment i.e., A string can only be replaced with a new string since its content can not be partially replaced.
- Strings are **immutable in python**.
- Consider the following example.
- Example 1

```
str = "HELLO"
```

```
str[0] = "h"
```

```
print(str)
```

output

Error

```
str = "HELLO"
```

```
print(str)
```

```
str = "hello"
```

```
print(str)
```

- **Output:**

HELLO

hello

Note:- the string str can be completely assigned to a new content as specified in the following example.

String Operators

- **+** It is known as concatenation operator used to join the strings given either side of the operator.
- ***** It is known as repetition operator. It concatenates the multiple copies of the same string.
- **[]** It is known as slice operator. It is used to access the sub-strings of a particular string.
- **[:]** It is known as range slice operator. It is used to access the characters from the specified range.
- **in** It is known as membership operator. It returns if a particular sub-string is present in the specified string.

- **not in** It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
- **r/R** It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
- **%** is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Example Program

```
str = "Hello"
```

```
str1 = " world"
```

```
print(str*3) # prints HelloHelloHello
```

```
print(str+str1)# prints Hello world
```

```
print(str[4]) # prints o
```

```
print(str[2:4]); # prints ll
```

```
print('w' in str) # prints false as w is not present in str
```

```
print('wo' not in str1) # prints false as wo is present in str1
```

```
.
```

```
print(r'C://python37') # prints C://python37 as it is written
```

```
print("The string str : %s"%(str)) # prints string str : Hello
```


Output:

HelloHelloHello

Hello world

o

ll

False

False

C://python37

The string str : Hello

String input

```
print("Enter your name:")
```

```
x = input()
```

```
print("Hello, " + x)
```

Number Input

```
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# Add two numbers
sum1 = int(num1)+int(num2)
prod1= int(num1)*int(num2)
print('The value of sum after addition: {} {}'.format(sum1,prod1))
```

Python Formatting operator

- Python provides an additional operator % which is used as an interface between the format specifiers and their values.
- In other words, we can say that it binds the format specifiers to the values.

Eg

Integer = 10;

Float = 1.290

String = "Ayush"

```
print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String));
```

Built-in String functions

- Python provides various in-built functions that are used for string handling.

1 . Python String Casefold() Method

- Python **Casefold()** method returns a lowercase copy of the string. It is more similar to lowercase method except it remove all case distinctions present in the string.

Example

```
str1 = "VIDYA ACADEMY"  
str2 = str.casefold() # Calling function  
# Displaying result  
print("Old value:", str1)  
print("New value:", str2)
```

2. Python String Center() Method

- Python **center()** method aligns string to the center by filling paddings left and right of the string.
- This method takes two parameters, first is a **width** and second is a **fillchar** which is optional. The fillchar is a character which is used to fill left and right padding of the string.

```
# Python center() function example
# Variable declaration
str1 = "Hello Python"
# Calling function
str2 = str1.center(20)
# Displaying result
print("Old value:", str1)
print("New value:", str2)
```

Output

Old value: Hello Python

New value: Hello Python

3. Python String Count() Method-It returns the number of occurrences of substring in the specified range

Syntax

`count(sub[, start[, end]])`

- It returns the number of occurrences of substring in the specified range. It takes three parameters, first is a substring, second a start index and third is last index of the range. Start and end both are optional whereas substring is required.

Python count() function example

Variable declaration

str1 = "Hello Python"

str2 = str.count('t')

Displaying result

print("occurences:", str2)

```
# Python count() function example
# Variable declaration
str1 = "ab bc ca de ed ad da ab bc ca"
oc = str1.count('a', 3)
# Displaying result
print("occurences:", oc)
```

- **Output:**
- occurrences: 5

```
# Python count() function example
# Variable declaration
str1 = "ab bc ca de ed ad da ab bc ca"
oc = str1.count('a', 3, 8)
# Displaying result
print("occurences:", oc)
```

- **Output:**
- occurrences: 1

```
# Python count() function example for count non-alphabet chars
# Variable declaration
str = "ab bc ca de ed ad da ab bc ca 12 23 35 62"
oc = str.count('2')
# Displaying result

print("occurences:", oc)
```

- **Output:**
- occurrences: 3

Python String `islower()` Method

- Python string `islower()` method returns True if all characters in the string are in lowercase. It returns False if not in lowercase.

Python `islower()` method example

Variable declaration

```
str = "hello python"
```

Calling function

```
str2 = str.islower()
```

Displaying result

```
print(str2)
```

Python String `isnumeric()` Method

- Python `isnumeric()` method checks whether all the characters of the string are numeric characters or not.
- It returns True if all the characters are true, otherwise returns False.

Python `isnumeric()` method example

Variable declaration

```
str = "12345"
```

Calling function

```
str2 = str.isnumeric()
```

Displaying result

```
print(str2)
```

Python isnumeric() method example

str = "123452500" # True

if str.isnumeric() == True:

print("Numeric")

else:

print("Not numeric")

str2 = "123-4525-00" # False

if str2.isnumeric() == True:

print("Numeric")

else:

print("Not numeric")

Output:

- Numeric
- Not numeric

- **Python String isupper() Method**
- Python **isupper()** method returns True if all characters in the string are in uppercase. It returns False if characters are not in uppercase.

Python isupper() method example

Variable declaration

```
str = "WELCOME TO PYTHON"
```

Calling function

```
str2 = str.isupper()
```

Displaying result

```
print(str2)
```

```
# Python isupper() method example
str = "WELCOME TO PYTHON"
str2 = str.isupper()
print(str2)
str3 = "WELCOME To JAVA."
str4 = str3.isupper()
print(str4)
str5 = "123 @$ -JAVA."
str6 = str5.isupper()
print(str6)
```

Python String upper() Method

- Python **upper()** method converts all the character to uppercase and returns a uppercase string.

```
# Python upper() method
```

```
# Declaring table and variables
```

```
str = "Hello java"
```

```
# Calling function
```

```
str2 = str.upper()
```

```
# Displaying result
```

```
print(str2)
```

Output:

HELLO JAVA

```
# Python upper() method
# Declaring variables
list = ["irfan","sohan","mohan"]
for l in list:
    print(l.upper()) # Displaying result
```

Output:

IRFAN

SOHAN

MOHAN

- An example that converts all the string into uppercase which starts with vowels. See the example below.

```
# Python upper() method
```

```
# Declaring variables
```

```
names = ["irfan","sohan","aman","mohan"]
```

```
vowels = ['a','e','i','o','u']
```

```
for l in names:
```

```
    for v in vowels:
```

```
        if(l.startswith(v)):
```

```
            print(l.upper()) # Displaying result
```

- **Output:**

- IRFAN

- AMAN

Python String swapcase() Method

- Python **swapcase() method** converts case of the string characters from uppercase to lowercase and vice versa.

```
# Python String swapcase() method
# Declaring variable
str = "HELLO JAVATPOINT"
# Calling function
str2 = str.swapcase()
# Displaying result
print (str2)
```

Output:

hello javatpoint