Chapter 2

Variables & Data Types

In this chapter, you will learn about

- Variables
- Variables Declaration
- Python Data Type 2.7
- Python Data Type 3.6
- Integer Data Type
- Literals
- Types of Literals
- Integer Literals Notations
- Python Valid Identifiers
- Real Numbers
- float Data Type
- Small Addition
- Boolean Data Type
- String
- String Literals
- String : Special Characters
- String : Slicing
- String : Immutable

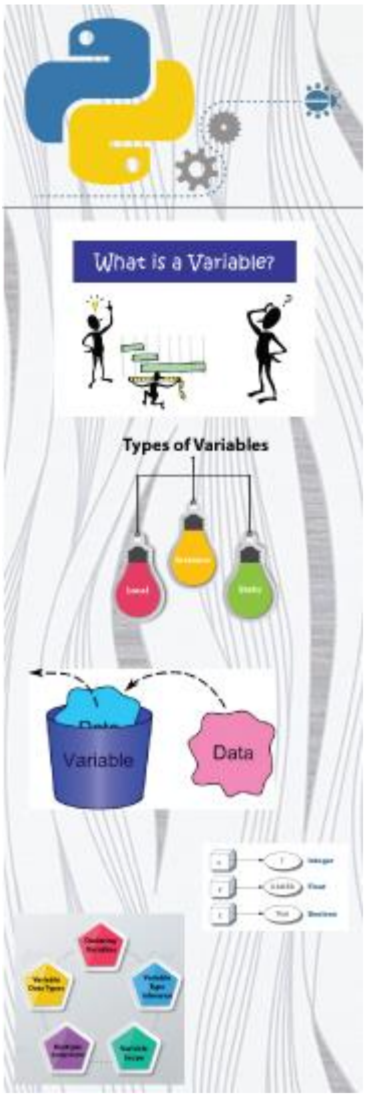


Table of Contents

❖ Variables

A variable is a symbolic name for a memory location in which data can be stored and subsequently recalled and modified. Variables are used for holding data values so that they can be utilized in various computations in a program.

A variable is an item of data named by an identifier. Variable declaration is manipulation of computer RAM's. We are reserving the space in the RAM and giving that space an easy-to-use name. So, in Python, if you declare a variable, you request the operating system to allocate a piece of memory. You (can) specify this piece of memory with a name and you can store some data in that piece of memory (for later use).

In Python, each variable has a specific data type that determines the storage size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The variable name is called as identifier.

In Python, every data variable is mapped to a unique memory address. In Python, a variable must be

⇒ **Declared (variable name) and**

⇒ **Initialized (value or reference assigned to a declared variable).**

Variable Declaration

Variable declaration is a Python statement. In Python, variables can be declared using the following syntax. All variable declaration has two important parts:

identifier = initial value

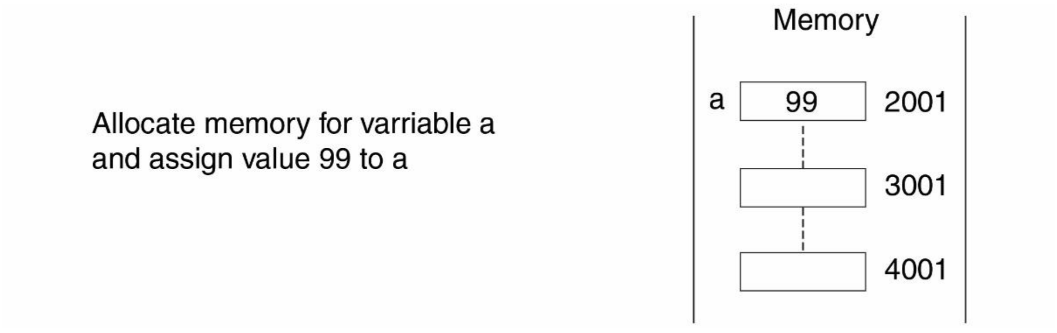
An **identifier** is used to refer unique memory address. You can give a variable any name you want, as long as it is a valid identifier.

The **initial value** can be any value.

For example,

```
a = 99
```

The a is the identifier or variable name. The value 99 is assigned to a.

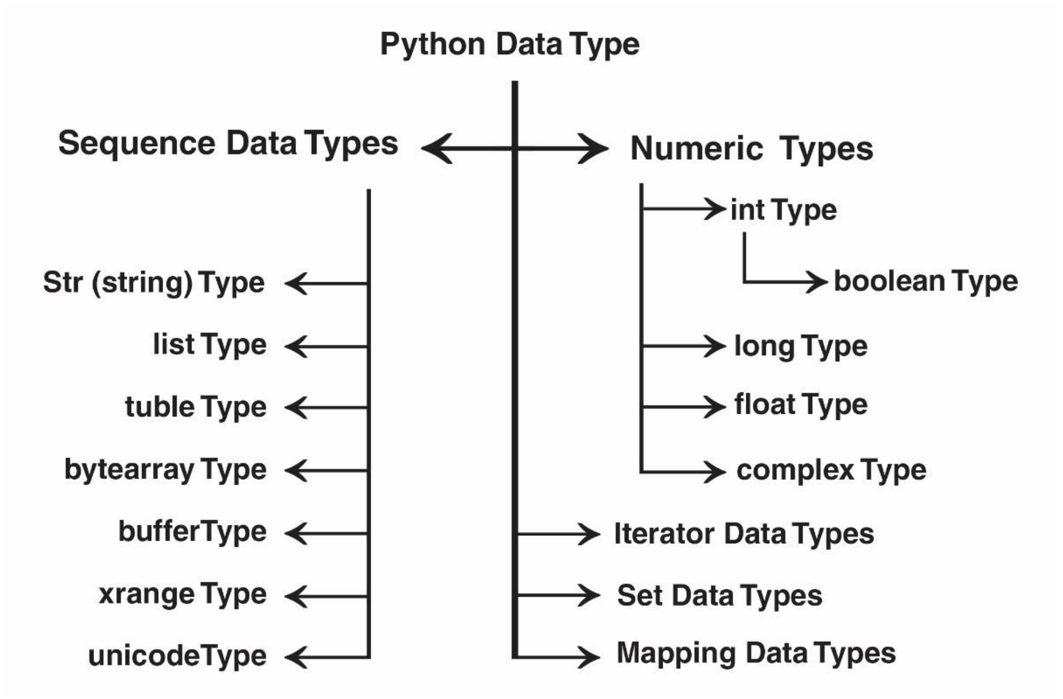


Variable Initialization

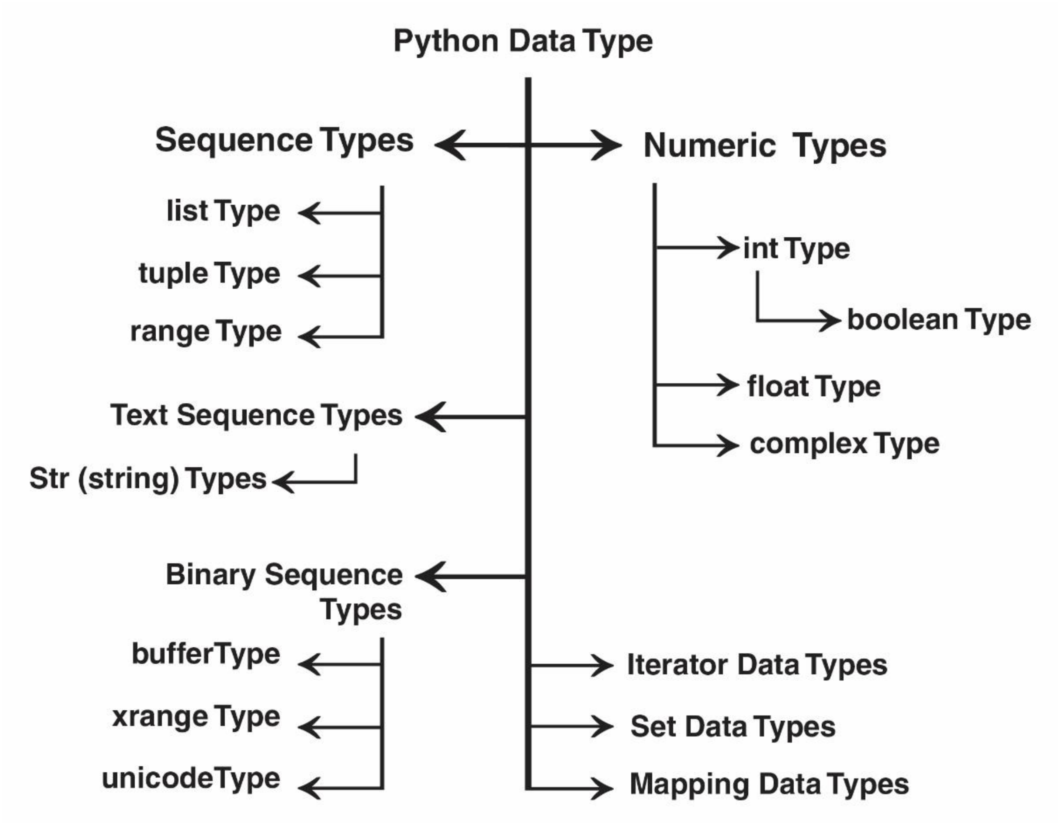
Variable Initialization is a Python statement, which assigns a value to a variable. Python supports only Manual Initialization. Programmer has to write the manual initialization. It does not support Automatic Initialization.

❖ Python Data Type

Python 2.7 Data Types



Python 3.6 Data Types



❖ Python Valid Identifiers

A valid identifier must a whole word, which contains a sequence of one or more letters, digits or underscores and the identifier must begin with a letter.

Upper and lowercase letters are distinct because identifier in Python is case sensitive. For instance, X is not equal to x.

Valid recommended identifiers

X

x

currentTax

_tax

Invalid Identifiers

1data	// error. First letter is a number
.val	// error. First letter is a dot

```
break              // error. Keyword
```

❖ Integers Primitive Value Data Types

Integers are those values which has no decimal part they can be positive or negative. like 12 or -12. Integer data types can store only values that are whole number. Integers are used to count discrete entities. We can have 5, 7, 9 humans, but we cannot have 4.22 humans. We can have 5.22 liters.

Python supports two kinds of Integers.

1. Signed Integers
2. Unsigned Integers

Type	Storage	Min Value	Max Value
Integer	Machine Capacity	Machine Capacity	Machine Capacity

Signed Integers

Signed Integers can interact with both positive and negative value.

Unsigned Integers

Unsigned Integers cannot handle negative values.

❖ Integer Representation

In computer, Integer values are represented in binary format. For example

iData = 1

The value 1 is represented as 0000 0000 0000 0000 0000 0001

iData = 5

The value 5 is represented as 0000 0000 0000 0000 0000 0101

iData = 22

The value 22 is represented as 0000 0000 0000 0000 0001 0110

Positive Values

Positive values are straightforward binary numbers.

The value 3 is represented as 0000 0000 0000 0000 0000 0011

The value 7 is represented as 0000 0000 0000 0000 0000 0111

The value 25 is represented as 0000 0000 0000 0000 0001 1001

Negative Values

Negative integers are represented using the two's complement method.

There are three steps involved to get the two's complement representation for a negative number,

1. Obtain the binary representation for the number's absolute value
2. Flip all the bits
3. Add 1.

For example, to represent byte value of -5

1.	Binary value is	0000 0000 0000 0000 0000 0101
2.	Flip all the bits	1111 1111 1111 1111 1111 1010
3.	Add 1	1111 1111 1111 1111 1111 1011

So, -5 represented as 1111 1111 1111 1111 1111 1011

❖ Literals

The term literal constant or literal refers to express a particular value that occurs within the source code of a program and cannot be changed. A literal constant is non-addressable, which means that its value is stored somewhere in memory, but we have no means of accessing that address.

Every literal has a value and a data type. The value of any literal does not change while the program runs and must be in the range of representable values for its type.

For example

```
x = 10
```

Here 10 is a literal constant.

❖ Type of Literals

Python supports five types of literals. They are

1. Integer Literal
2. Float Literal
3. Complex Number Literal

4. String Literal
5. Bytes Literal

❖ Integer Literals

Integer Literals Notations

In Python, Integer Literals can be specified in three different notations:

1. Integer Literal (base 10)
2. Octal (base 8)
3. Hexadecimal (base 16)

Integer Literals

Integer Literals are represented in base 10.

Python does not allow you prefix 0 with integer literals.

Octal Literals

Octal Literals are denoted in Python by a leading 0o or 0O (Zero and Alphabetic o or O). For example

```
print(0O7)      # It prints 7
print(0o10)     # It prints 8
```

Hexadecimal Literals

Hexadecimal numbers are preceded with 0x or 0X characters. A prefix specifies the base or radix: 0x or 0X for hexadecimal, and nothing for decimal.

The possible hex digits are

0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

For example

```
print(0x7)      # It prints 7
print(0XA);     # It prints 10
```

❖ Float Data Type

Python language supports one floating point types:

1. float,

It is based on the IEEE 754 standard.

float Data Type

The float data type stores the floating point values in the form of +/ – m × 2e.

- m indicates Mantissa
- e indicates Exponent

In Python, a floating point number is accurate up to 15 decimal places. To differentiate, Integer and floating points are separated by decimal points. 99 is integer, 99.0 is floating point number.

❖ Small Addition Problem

```
1. _sum = 0.0
2. for i in range(10):
3.      print(i)
4.      _sum += 0.1
5. print(_sum)
```

Theoretically, the Line 5, has to print sum as 1.0. But, it prints sum Value Is 0.9999999999999999. The small error in representing in binary propagates to the sum.

❖ Boolean Data Type

In Python, Boolean variables can hold only one of two possible values:

1. true
2. false

The type for a logical variable is boolean, named after George Boole, who developed Boolean algebra.

In Python, the data type boolean is considered as an integer type.

Boolean variables are also referred to as **logical variables**. Variables of type bool are used to store the results of tests that can be either true or false, such as whether one value is equal to another.

There are two boolean literal values:

1. true (1)
2. false (0).

Of course, you can also initialize variables of type boolean when you declare them:

isAvailable = true

❖ String Data Type

Python can manipulate Strings. Strings is a series of characters gather together like

- Empty String
- Blank Character
- Single Character
- Word
- Phrase
- Any combination of characters.

String Literals

Zero or more series of characters enclosed between quotes are called String Literals.

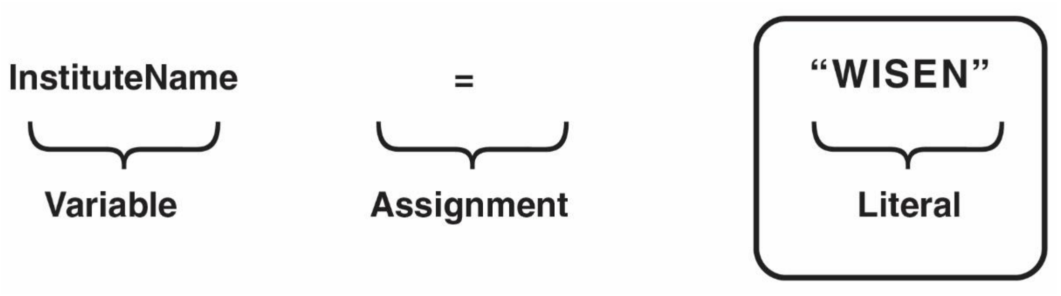
In Python, Strings is enclosed in

1. Single Quote or

   print('Hello WISEN')

2. Double Quote

   print("Hello WISEN")


$$\underbrace{\text{InstituteName}}_{\text{Variable}} \quad \underbrace{=}_{\text{Assignment}} \quad \boxed{\underbrace{\text{“WISEN”}}_{\text{Literal}}}$$

❖ String Literals : Escape Quotes

- When a string enclosed in Double Quotes, you can have Single Quote within the string.

  print("Let's achieve Our Dream Career")

- When a string enclosed in Double Quotes, you can't have Double Quote within the string.

  print('Let"s achieve Our Dream Career')

- When a string enclosed in Double Quotes, you can't have Double Quote within the string.

  print("Let"s achieve Our Dream Career")

```
print("Let"s achieve Our Dream Career")
                   ^
SyntaxError: invalid syntax
```

To overcome the above Syntax Error, the Escape Quote \ can be prefaced as below

print("Let\"s achieve Our Dream Career")

- When a string enclosed in Single Quotes, you can't have Single Quote within the string.

```
print('Let's achieve Our Dream Career')
                   ^
SyntaxError: invalid syntax
```

To overcome the above Syntax Error, the Escape Quote \ can be prefaced as below

print('Let\'s achieve Our Dream Career')

❖ Special Characters

In Python, some strings are considered as special characters. One or more characters that starts with a backslash is known as an escape sequence.

The below table shows some of the most commonly used escape sequences in Python.

Special Character	Meaning
\\	It indicates backslash (\)
\’	It indicates single quote (')
\”	It indicates double quote (")
\a	It indicates ASCII Bell (BEL)
\t	It indicates ASCII horizontal Tab (TAB)
\n	It indicates newline (linefeed)
\r	It indicates ASCII carriage return (CR)

For example, without raw string

```
1. print("Mary \n\t Brown")
```

Line 1, prints the below output in the console

```
Mary
	Brown
```

❖ Raw Strings

The String prefixed with r or R are called raw strings. To specify as a raw string, prefix an r or R character before the opening single quote ' or double quote " character that encloses the string. It applies different rules for backslash escape characters.

If the string prefix with r or R and contains backslash within the string, Python will not interpret these backslashes as escape sequences.

For example, without raw string

```
1. print("Mary \n\t Brown")
```

Line 1, prints the below output in the console

Mary

    Brown

For example, with raw string

```
print(r"Mary \n\t Brown")
```

Line 1, prints the below output in the console

Mary \n\t Brown

❖ Multi Line String Literals

In Python, String literals can span multiple lines. This is implemented using triple quoted string literal format. It is also called block string. The quote can be a single quote or double quote.

This form begins with three quotes is followed by number of lines of text closed with the same triple quote sequence which is opened with.

The general form is

```
“”” ………..
…………….
…………….
……………. “””
```

Or

```
‘’’ ………..
…………….
…………….
……………. ‘’’
```

Any characters between the starting and ending quotes becomes part of the string. This string will also contain both blanks and newlines.

For example

```
1. print("""
                WISEN
                  IT
                Solutions
                """)
2. print('''
                Chennai
                  India
                   ''')
```

Line 1, after the triple quotes, Enter has been pressed. Therefore, WISEN has been displayed in the next line.

Next one new line is added.

Line 2, after the triple quotes, Enter has been pressed. Therefore, Chennai has been displayed in the next line.

The final output will be

```
WISEN

IT

SOLUTIONS


Chennai

    India
```

There is a line between SOLUTIONS & Chennai.

New Lines are automatically included in the string. You can prevent this new line by adding a \ at the beginning of the line.

For example

```
print("""
                WISEN
                  IT
                Solutions
                """)
print('''\
                Chennai
                  India
                   ''')
```

The new line is not added at the beginning of the line, since \ (backward slash) is specified at the beginning of the string.

❖ String Concatenation

In Python, String Concatenation can be performed in two ways:

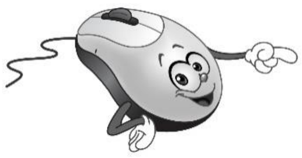
1. Using + Operator
2. Literals Next to Each Other

Using + Operator

Strings can be concatenated (glued together) with the + operator. The + Operator is used to concatenate variables or a variable and a literals.

```
firstName = "Mary"
lastName = “Brown”
fullName = firstName + lastName
```

Line 3, the + operator concatenates firstName and lastName and assign to fullName variable using = operator. Therefore, the fullName contains MaryBrown.



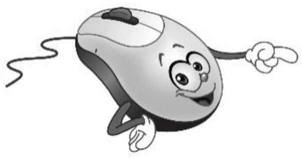
In Python, the + operator cannot be used to concatenate two different data type values. For example, Python does not allow to concatenate string and integer value.

Literals Next to Each Other

In Python, two or more string literals next to each other are automatically concatenated.

```
1. result = 'WISEN ' 'IT '          "SOLUTIONS"
```

Line 1, there is one or more space between WISEN & IT & SOLUTIONS literals. Therefore, Python concatenates all three results and assign it to result variable. The result variable will contain WISEN IT SOLUTIONS



String Literals next to each other are automatically concatenated works only with literals.

Not with Variables and Expressions.

❖ String Repetitions

In Python, String Repetitions can be performed in one way:

1. Using * Repletion Operator

This Repetition Operator repeats a string with a fixed number of times.

```
1. word = "ha "
2. print(word * 5)
```

The line 2 prints ha ha ha ha ha

❖ Character Retrieval

Python allows you determines only one character at the given index within the string.

In Python, the index of the first character is 0.

In Python, the index of last character is **no of character in string – 1.**

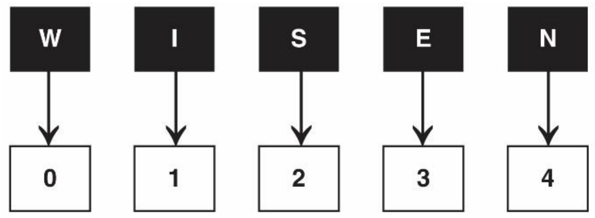
Character Retrieval

In Python, the character can be retrieved using [ ] bracket.

The general form is

variable[index] or stringLiteral[]

For example



```
instituteName = "WISEN"
print(instituteName[0])         # It prints W
print(instituteName[3])         # It prints E
print(instituteName[4])         # It prints N
```

Character Retrieval : Reverse Order

In Python String indexes can be negative numbers.

The count starts from right to left.

In Python, the index of the last character is -1.

In Python, the index of first character is **no of character in string**

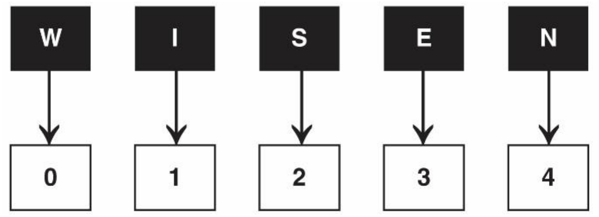
Character Retrieval : Reverse Order

In Python, the character can be retrieved using [ ] bracket.

The general form is

variable[index] or stringLiteral[]

For example



```
instituteName = "WISEN"
print(instituteName[-1])        # It prints N
print(instituteName[-2])        # It prints E
print(instituteName[-5])        # It prints W
```

Character Retrieval : Out of Range

In Python, when you try to retrieve a character which out of range causes an Index Error.

For Example

```
instituteName = "WISEN"
print(instituteName[5])
```

In Line 2, Python generates the Index Error, since you are trying retrieve a character in index position 5; but the last index number is 4. Therefore you are accessing the out of range index number.

```
Line 2

print(instituteName[5])

IndexError: string index out of range
```

❖ Character Slicing

String : Slicing

Python supports slicing. Slicing allows you to determine substring. Substring means zero or more characters.

The general form is

variable[fromIndex : toIndex]

or

stringLiteral[fromIndex : toIndex]

For example

```
1. instituteName = "WISEN"

2. instituteName[1:4]

   # variable[fromIndex:toIndex] Format

3. "India"[1:4]

   # stringLiteral [fromIndex:toIndex] Format
```

The fromIndex is **inclusive**. The toIndex is **exclusive**.

```
instituteName = "WISEN"
print(instituteName[1:4])       # It prints ISE
print("India"[1:4])             # It prints ndi
```

String Slicing : Indexes are Optional

The fromIndex or toIndex can be omitted. But both cannot be omitted.

If fromIdex omitted, the general form is

variable[: toIndex]

or

stringLiteral[: toIndex]

If toIndex omitted, the general form is

variable[fromIndex :]

or

stringLiteral[fromIndex :]

If fromIndex is omitted, it defaults to zero (0).

If toIndex is omitted, it defaults to string no of characters.

```
instituteName = "WISEN"
print(instituteName[:4])        # It prints WISE
print(instituteName[1:])        # It prints ndi
```

String : Slicing : Negative Indexes

Python supports negative indexes on slicing operations.

The general form is

variable[-fromIndex : -toIndex]

or

stringLiteral[-fromIndex : -toIndex]

```
instituteName = "WISEN"
print(instituteName[-5:-2])     # It prints WIS
print(“India”[-4:-2])           # It prints nd
```

String Slicing : Negative Indexes are Optional

The fromIndex or toIndex can be omitted. But both cannot be omitted.

If fromIdex omitted, the general form is

variable[: -toIndex]

or

stringLiteral[: -toIndex]

If toIndex omitted, the general form is

variable[-fromIndex :]

or

stringLiteral[-fromIndex :]

If fromIndex is omitted, it defaults to –no of characters.

If toIndex is omitted, it defaults to 0.

```
instituteName = "WISEN"
print(instituteName[:-2])       # It prints WIS
print(instituteName [-4:])      # It prints ISEN
```

String : Slicing : Out of Range

While slicing, if the indexes are out of range, python will not throw any errors.

```
1. country = "India"

2. print(country[3:10])      # It prints ia

3. print(country[5:10])

   # It prints “” (Empty String)
```

In the above code snippet,

Line 2, the toIndex is out of range. Even though Python does not throw any error.

Line 3, the fromIndex and toIndex both are out of range. Even though Python does not throw any error.

❖ String Length

The built-in function len() determines no of characters in the string. The general form is

len(variable or StringLiteral)

```
country = "India"

print(len(country))       # It prints 5

print(len(“Wisen”)        # It prints 5
```

❖ String Are Immutable

Strings are immutable. Therefore, Python does not allow you modify or assign an individual character to an indexed position of a String.

```
sData = "MyValue"

sData[2] = ‘A’      # Syntax Error
```

```
Line 2

 sData[2] = "A"

TypeError: 'str' object does not support
item assignment
```

❖ Summary

- A variable is a **symbolic name for a memory location** in which **data can be stored**.
- In Python, you have **only variable with assignment**.
- Integer data types can **store only values that are whole number**.
- Python supports five types of literals. They are
  1. Integer Literal
  2. Float Literal
  3. Complex Number Literal
  4. String Literal
  5. Bytes Literal
- In Python, Integer Literals can be specified in three different notations:
  1. Integer Literal (base 10)
  2. Octal (base 8)
  3. Hexadecimal (base 16)
- Octal Literals are denoted in Python by a **leading 0o or 0O.**
- Hexadecimal numbers are **preceded with 0x or 0X characters.**
- A valid identifier must a

- ✓ whole word, which contains a
- ✓ sequence of one or more letters,
- ✓ digits or underscores and
- ✓ identifier must begin with a letter.

- Upper and lowercase letters are distinct because identifier in Python is case sensitive
- Python language supports one floating point types: **float**
- In Python, Boolean variables can hold only one of two possible values:

  true and **false**.
- Python can **manipulate Strings**. Strings is a **series of characters gather together.**
- **Zero or more series of characters** enclosed between quotes are called String Literals**.**
- In Python, **some strings** are **considered as special characters**.
- **The String prefixed with r or R** are called **raw strings**.
- In Python, **String literals can span multiple lines**. This is **implemented using triple quotes**.
- In Python, String Concatenation can be performed in two ways:
  1. Using + Operator
  2. Literals Next to Each Other
- In Python, **String Repetitions** can be performed in one way: **Using * Operator**
- Python allows you **determines only one character at the given index** within the string.
- In Python, the **index of the first character is 0**. In Python, the **index of last character** is **no of character in string - 1**
- Python **supports slicing**. Slicing allows you to **determine substring**. The built-in function len() determines no of characters in the string. In Python, String are immutable.
- Therefore, you can't assign value to an indexed position in the string. Substring means **zero or more characters**.