

PYTHON – REVISION TOUR - I

- ✓ INTRODUCTION
- ✓ TOKENS
- ✓ BAREBONES OF PYTHON
- ✓ VARIABLES AND ASSIGNMENT
- ✓ INPUT & OUTPUT
- ✓ DATA TYPES
- ✓ MUTABLE AND IMMUTABLE TYPESS
- ✓ EXPRESSION
- ✓ FLOW OF CONTROL
- ✓ JUMP STATEMENT

Introduction

- ❖ Python was created by Guido Van Rossum
- ❖ The language was released in February 1991
- ❖ Python got its name from a BBC comedy series from seventies-
“Monty Python”s Flying Circus”
- ❖ Python can be used to follow both Procedural approach and
Object Oriented approach of programming
- ❖ It is free to use
- ❖ Python is based on or influenced with two programming
languages:
 - ❖ ABC language [replacement of BASIC]
 - ❖ Modula-3

Features which make Python so popular

- ❖ Easy to use Object oriented language
- ❖ Expressive language
- ❖ Interpreted Language
- ❖ Its completeness
- ❖ Cross-platform Language
- ❖ Free and Open source
- ❖ Variety of Usage / Applications

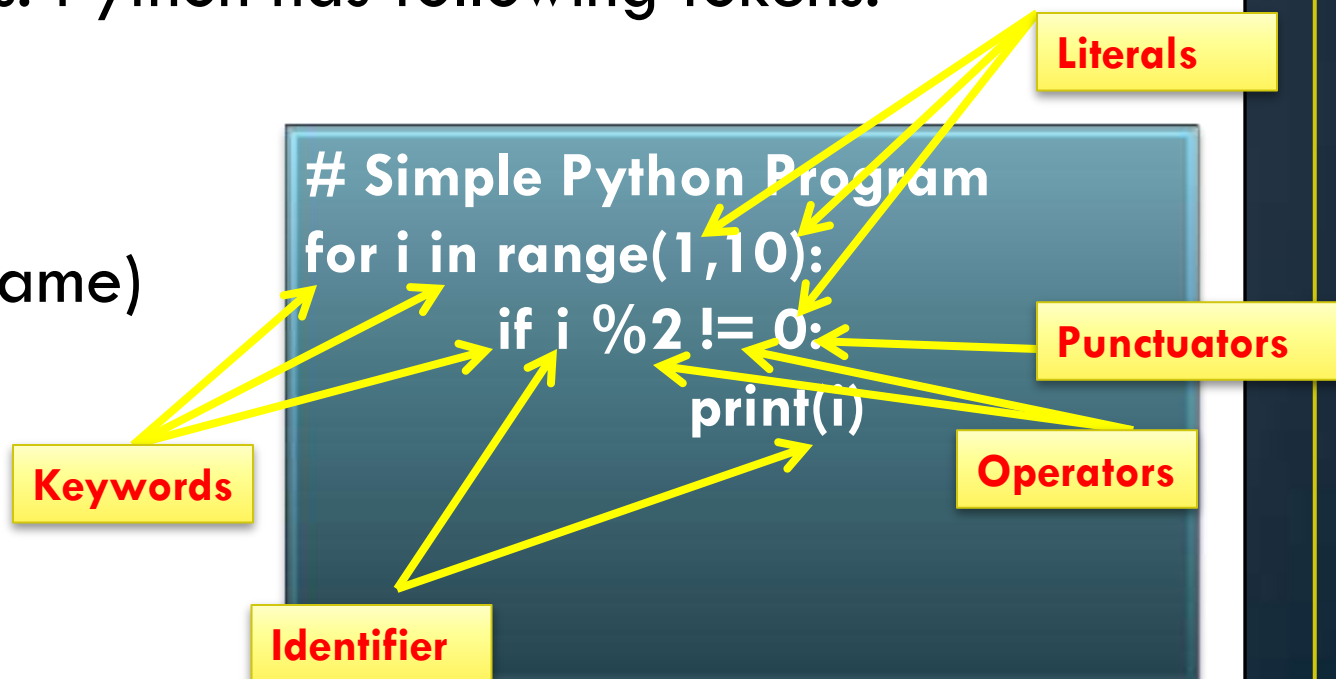
Limitations (Minus) of Python

- ❖ Not the fastest language
- ❖ Lesser Libraries than C, Java, Perl
- ❖ Not Strong on Type-binding
- ❖ Not Easily convertible

TOKENS

In a passage of text, individual words and punctuation marks are called tokens or lexical units or lexical elements. The smallest individual unit in a program is known as Tokens. Python has following tokens:

- ✓ Keywords
- ✓ Identifiers(Name)
- ✓ Literals
- ✓ Operators
- ✓ Punctuators



KEYWORDS

Keywords are the reserved words and have special meaning for python interpreter. Every keyword is assigned specific work and it can be used only for that purpose.

A partial list of keywords in Python is

and	del	from	not
while	as	elif	global
or	with	assert	else
if	pass	yield	break
except	import	print	class
exec	in	raise	continue
finally	is	return	def
for	lambda	try	

IDENTIFIERS

Are the names given to different parts of program like variables, objects, classes, functions etc.

Identifier forming rules of Python are :

- Is an arbitrarily long sequence of letters and digits
- The first character must be letter or underscore
- Upper and lower case are different
- The digits 0-9 are allowed except for first character
- It must not be a keyword
- No special characters are allowed other than underscore is allowed.
- Space not allowed

The following are some valid identifiers

GradePay
GRADEPAY

File_12_2018
_ismarried

JAMES007
_to_update

The following are some invalid identifiers

Grade-Pay
if

12_2018_File
RollNo.

\$JAMES007
Roll No

Literals / Values

- Literals are data items that have a fixed value. Python supports several kinds of literals:
 - String Literal
 - Numeric Literals
 - Boolean Literals
 - Special Literals – **None**
 - Literal Collections

String Literals

- It is a collection of character(s) enclosed in a double or single quotes. It can be either Single line strings or Multiline Strings
- **SINGLE LINE STRING** : *must terminate in one line i.e. the closing quotes should be on the same line as that of the opening quotes*
- *Examples of String literals*
 - “Python”
 - “Mogambo”
 - ‘123456’
 - ‘Hello How are your’
 - ‘\$, ‘4’, ”@@”
- In Python both single character or multiple characters enclosed in quotes such as “kv”, ‘kv’, ’*’, ”+” are treated as same

String Literals

- **MULTI LINE STRING** : To store multiline string Python provides two ways:
- (1) *By adding a backslash at the end of normal Single / Double quoted string. For e.g.*

```
>>> Name="1/6 Mall Road \
Kanpur"
```

```
>>> Name
'1/6 Mall RoadKanpur'
>>>
```

- (2) *By typing text in triple quotation marks*

for e.g.

```
>>> Address="""1/7 Preet Vihar
New Delhi
India"""
```

```
>>> print(Address)
```

```
1/7 Preet Vihar
New Delhi
India
```

```
>>> Address
```

```
'1/7 Preet Vihar\nNew Delhi\nIndia'
```

List of Escape characters

Escape Sequence	What it does	Escape Sequence	What it does
\\	Backslash	\r	Carriage return
\'	Single quotes	\t	Horizontal tab
\"	Double quotes	\uxxxx	Hexadecimal value(16 bit)
\a	ASCII bell	\Uxxxx	Hexadecimal value(32 bit)
\b	Back Space	\v	vertical tab
\n	New line	\ooo	Octal value

Size of String

- Python determines the size of string as the **count of characters** in the string. For example size of **string “xyz” is 3** and of **“welcome” is 7**. But if your string literal has an escape sequence contained in it then make sure to count the **escape sequence as one character**. For e.g.

String	Size
'\\'	1
'abc'	3
'\ab'	2
“Meera\'s Toy”	11
“Vicky's”	7

- You can check these size using **len() function of Python**. For example
- >>>len('abc')** and press enter, it will show the size as 3

Numeric Literals

- The numeric literals in Python can belong to any of the following numerical types:

1) INTEGER LITERALS: it contain at least one digit and must not contain decimal point. It may contain (+) or (-) sign.

- Types of Integer Literals:

a) Decimal : 1234, -50, +100

b) Octal : it starts from symbol **0o (zero followed by letter 'o')**

- For e.g. **0o10** represent decimal 8

Numeric Literals

```
>>> num = 0o10
```

```
>>> print(num)
```

It will print the value 8

c) Hexadecimal : it starts from **0x (zero followed by letter 'x')**

```
>>> num = 0xF
```

```
>>> print(num)
```

it will print the value 15

```
>>> num = 0xABC
```

```
>>> print(num)
```

it will print the value 2748

Numeric Literals

- 2) Floating point Literals: also known as real literals. Real literals are numbers having fractional parts. It is represented in two forms Fractional Form or Exponent Form
 - Fractional Form: it is signed or unsigned with decimal point
 - For e.g. 12.0, -15.86, 0.5, 10. (will represent 10.0)
 - Exponent Part: it consists of two parts “**Mantissa**” and “**Exponent**”.
 - For e.g. 10.5 can be represented as $0.105 \times 10^2 = 0.105E02$ where 0.105 is mantissa and 02 (after letter E) is exponent
- 3) Complex number Literals : Complex number in python is made up of two floating

Example

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

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

```
1.0 0.0
```


Boolean Literals

A Boolean literals in Python is used to represent one of the two Boolean values i.e. **True or False**

These are the only two values supported for Boolean Literals

For e.g.

```
>>> isMarried=True
```

```
>>> type(isMarried)
```

```
<class 'bool'>
```

Special Literals None

Python has one special literal, which is None. It indicate absence of value. In other languages it is knows as **NULL**. It is also used to indicate the end of lists in Python.

```
>>> salary=None
```

```
>>> type(salary)
```

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

Operators

- are symbol that perform specific operation when applied on variables. Take a look at the expression:

(Operator)
10 + 25 \longrightarrow **(Operands)**

Above statement is an expression (combination of operator and operands)

i.e. operator operates on operand. some operator requires two operand and some requires only one operand to operate

Operators

- Operators can be

Type of Operators	Symbols
Arithmetic	+, -, *, /, %, **, //
Relational	>, <, >=, <=, ==, !=
Logical	and, or
Identity	is, is not
Assignment	=
Membership	in, not in
Arithmetic Assignment	+=, -=, *=, /=, %=, **=, //=
Bitwise	&, ^, , <<, >>

Punctuators

- Punctuators are symbols that are used in programming languages to organize sentence structure, and indicate the rhythm and emphasis of expressions, statements, and program structure.
- **Common punctuators are: ' " # \$ @ [] {} = :: ; () , .**

Barebones of Python Program

- It means basic structure of a Python program
- Take a look of following code:

```

#This program shows a program's component
# Definition of function SeeYou() follows
def SeeYou():
    print("This is my function")
#Main program
A=10
B=A+20
C=A+B
if(C>=100)
    print("Value is equals or more than 100")
else:
    print("Value is less than 100")
SeeYou()
    
```

Comments
 Function
 Expressions
 Inline Comment
 Block
 #checking condition
 #Calling Function

Statements

Indentation

Variables

- Variables are named temporary location used to store values which can be further used in calculations, printing result etc. Every variable must have its own Identity, type and value. Variable in python is created by simply assigning value of desired type to them.
- **For e.g**
- **Num = 100**
- **Name="James"**

Variables

- **Note: Python variables are not storage containers like other programming language. Let us analyze by example.**
- **In C++, if we declare a variable radius:**

radius = 100

[suppose memory address is 41260]

Now we again assign new value to radius

radius = 500

Now the memory address will be still same only value will change

Variables

- *Now let us take example of Python:*

radius = 100 [memory address 3568]

radius = 700 [memory address 8546]

Now you can see that In python, each time you assign new value to variable it will not use the same memory address and new memory will be assigned to variable. In python the location they refer to changes every time their value change.(This rule is not for all types of variables)

Dynamic Typing

- In Python, a variable declared as numeric type can be further used to store string type or another.
- Dynamic typing means a variable pointing to a value of certain type can be made to point to value/object of different type.
- Lets us understand with example

```
x = 100          # numeric type
```

```
print(x)
```

```
x="KV OEF"      # now x point to string type
```

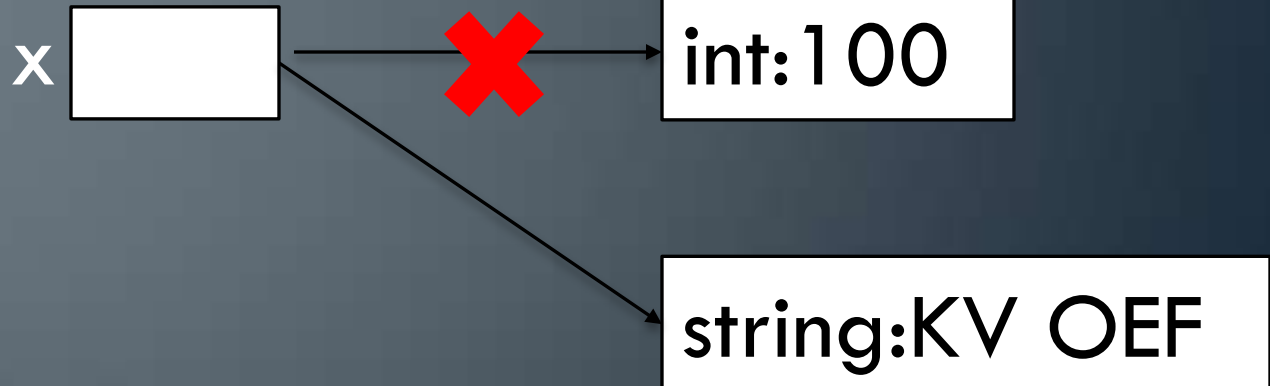
```
print(x)
```

Dynamic Typing

x=100



x="KV OEF"



Caution with Dynamic Typing

- Always ensure correct operation during dynamic typing. If types are not used correctly Python may raise an error.
- Take an example

```
x = 100
```

```
y = 0
```

```
y = x / 2
```

```
print(y)
```

```
x='Exam'
```

```
y = x / 2 # Error, you cannot divide string
```

Static Typing Vs Dynamic Typing

- Dynamic typing is different from Static typing. In Static Typing, a data type is attached with a variable when it is defined first and it is fixed. That is data type of variable cannot be changed in Static typing whereas there is no such restriction in dynamic typing, which is supported by Python.

Multiple Assignments

- Python is very versatile with assignments. Let's see in how different ways we can use assignment in Python:

1. **Assigning same value to multiple variable**

`a = b = c = 50`

2. **Assigning multiple values to multiple variable**

`a,b,c = 11,22,33`

Note: While assigning values through multiple assignment, remember that Python first evaluates the RHS and then assigns them to LHS

Multiple Assignments

```
x,y,z = 10,20,30           #Statement 1  
z,y,x = x+1,z+10,y-10    #Statement 2  
print(x,y,z)
```

Output will be

10 40 11

Now guess the output of following code fragment

```
x,y = 7,9  
y,z = x-2, x+10  
print(x,y,z)
```

Multiple Assignments

Let us take another example

```
y, y = 10, 20
```

*In above code first it will assign 10 to y and again it assign 20 to y, so if you print the value of y it will print **20***

Now guess the output of following code

```
x, x = 100, 200
```

```
y, y = x + 100, x + 200
```

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


Simple Input and Output

- In python we can take input from user using the built-in function `input()`.
- Syntax

```
variable = input(<message to display>)
```

Note: value taken by `input()` function will always be of String type, so by default you will not be able to perform any arithmetic operation on variable.

```
>>> marks=input("Enter your marks ")
```

```
Enter your marks 100
```

```
>>> type(marks)
```

```
<class 'str'>
```

Here you can see even we are entering value 100 but it will be treated as string and will not allow any arithmetic operation.

Simple Input and Output

```
>>> salary=input("Enter your salary ")
```

```
Enter your salary 5000
```

```
>>> bonus = salary*20/100
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Reading / Input of Numbers

- Now we are aware that `input()` function value will always be of string type, but what to do if we want number to be entered. **The solution to this problem is to convert values of `input()` to numeric type using `int()` or `float()` function.**

```
>>> age = int(input("Enter your age "))
```

```
>>> print type(age)
```

```
<type 'int'>
```

```
>>> salary = float(input("Enter salary"))
```

```
>>> print type(salary)
```

```
<type 'float'>
```

Output through print()

- Python allows to display output using print().
- Syntax:

```
print(*Object [,sep="string",end="string"])
```

here *Object means one or multiple comma separated objects/messages to be printed. It convert everything (*Object) in String before printing

Example 1

```
print("Welcome")
```

Example 2

```
print(100)
```

Example 3

```
Age=20
```

```
print("Your age is ", Age)
```

Output through print()

Example 4

```
r = int(input("Enter Radius "))  
print("Area of circle is ",3.14*r*r)
```

Example 5

```
print('Amar','Akbar','Anthony')
```

Output will be : **Amar Akbar Anthony**

space will be automatically inserted between different values as separator because the default value of 'sep' parameter is space

Example 6

```
Print('Amar','Akbar','Anthony', sep='**')
```

Output will be : **Amar**Akbar**Anthony**

Output through print()

Note:

By default each print statement appends a new line character after the printed value. The default value of 'end' parameter is “\n”

Example

```
print("Learning Python")
```

```
print("Developed by Guido Van Rossum")
```

Output

Learning Python

Developed by Guido Van Rossum

Output through print()

Note:

We can change the value of end to any other value.

Example

```
print("Learning Python ",end="😊")
```

```
print(" Developed by Guido Van Rossum")
```

Output

Learning Python 😊 Developed by Guido Van Rossum

Let us Recall Program 1

Open a new script file and type the following code:

```
num1=int(input("Enter Number 1 "))
num2=int(input("Enter Number 2 "))
num3 = num1 + num2
print("Result =",num3)
```

Save and execute by F5 and observe the result

Let us write few programs

- WAP to calculate perimeter of rectangle
- WAP to enter radius and calculate area of circle
- WAP to enter Name, marks of 5 subject and calculate total & percentage of student
- WAP to enter distance in feet and convert it into inches
- WAP to enter value of temperature in Fahrenheit and convert it into Celsius.
- WAP to enter radius and height of cylinder and calculate volume of cylinder.

DATA TYPES

- Data type in Python specifies the type of data we are going to store in any variable, the amount of memory it will take and type of operation we can perform on a variable. Data can be of many types e.g. character, integer, real, string etc.
- **Python supports following data types:**
 - Numbers (int, float, complex)
 - String
 - List
 - Tuple
 - Dictionary

Numbers

- From the name it is very clear the Number data types are used to store numeric values. Numbers in Python can be of following types:
 - (i) Integers
 - a) Integers(signed)
 - b) Booleans
 - (ii) Floating point numbers
 - (iii) Complex Numbers

Integers

- Integers allows to store whole numbers only and there is no fraction parts. Integers can be positive and negative e.g. 100, 250, -12, +50
- There are two integers in Python:
 - 1) **Integers(signed)** : it is normal integer representation of whole numbers. Integers in python can be on any length, it is only limited by memory available. In Python 3.x int data type can be used to store big or small integer value whether it is +ve or -ve.
 - 2) **Booleans**: it allows to store only two values **True** and **False**. The internal value of boolean value True and False is 1 and 0 resp. We can get boolean value from 0 and 1 using bool() function.

Floating Point Numbers

- Floating point numbers are mainly used for storing values like distance, area, temperature etc. which have a fractional part.
- Floating point numbers have two advantages over integers:
 - ✓ they can represent values between the integers
 - ✓ they can represent a much greater range of values
- But floating point numbers suffer from one disadvantage also:
 - ✓ Floating point operations are usually slower than integer operations.
- *In Python floating point numbers represent machine level double precision floating point numbers i.e. 15 digit precision.*

Complex Numbers

- Python represent complex numbers in the form $A+Bj$. To represent imaginary numbers, Python uses j or J in place of i . So in Python $j = \sqrt{-1}$. Both real and imaginary parts are of type float

e.g.

$$a = 0 + 6j$$

$$b = 2.5 + 3J$$

```
>>>a=4+5j
```

```
>>>print(a)           # (4+5j)
```

```
>>>b=0+2j
```

```
>>>b                   #(2j)
```

Complex Numbers

- Python allows to retrieve real and imaginary part of complex number using attributes: **real** and **imag**
- If the complex number is a then we can write a.real or a.imag
- Example
- ```
>>>a=1+3.54j
```
- ```
>>>print(a.real)           # 1.0
```
- ```
>>>print(a.imag) # 3.54
```

# String Data Type

- All string in Python is a sequence of Unicode characters. Unicode supports every characters from every language.
- Following are all legal strings in Python
  - “LEARNING”, “135”, “\$\$##”, “भारत”,
  - *Every character in String is at a particular position called INDEX, which starts from 0 (zero) i.e. first character will be at INDEX 0 and so on..*



# String

- In Python string is a sequence of characters and each character can be individually access using index. From beginning the first character in String is at index 0 and last will be at len-1. From backward direction last character will be at index -1 and first character will be at -len.

Forward indexing

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
| W  | E  | L  | C  | O  | M  | E  |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Backward indexing

# List Data Type

- 1. A list in python represents a list of comma-separated values of any data type between square brackets
- 2. It is mutable i.e. values are changeable
- [10,20,30,40,50]
- ['a','e','o','i','u']
- ["KV",208004,97.5]
- **Example :**
- >>> family=["Mom","Dad","Sis","Bro"]
- >>> print(family)
- ['Mom', 'Dad', 'Sis', 'Bro']
- >>> 'Tommy' in family # Output will be false

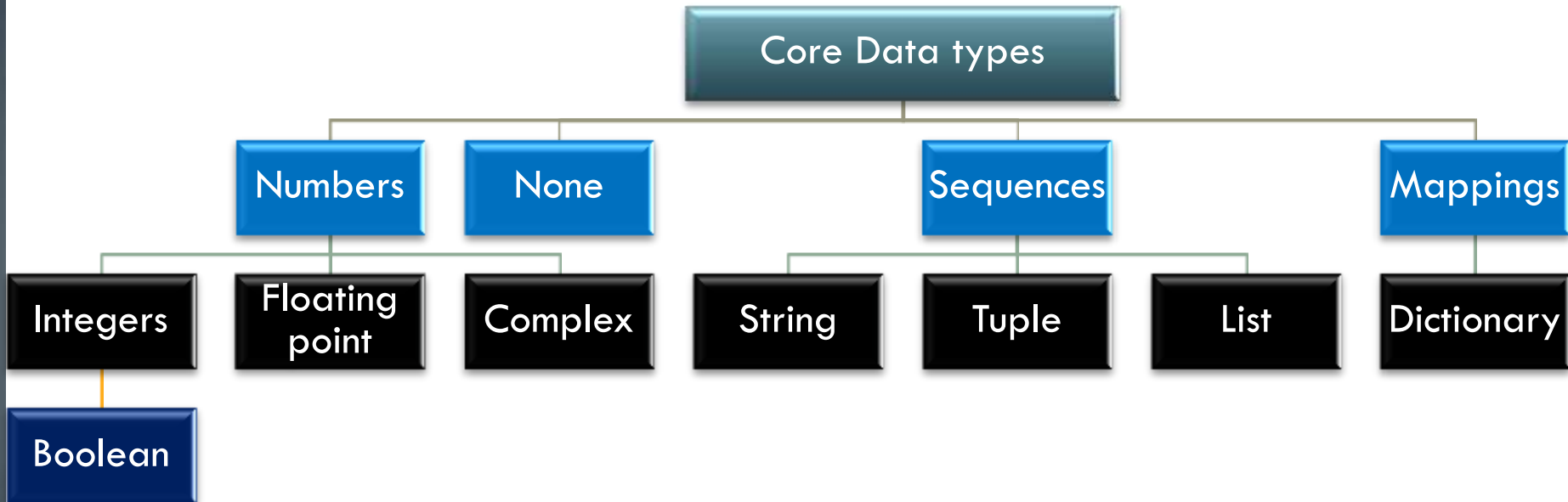
# Tuple Data Type

- Tuples are those lists which cannot be changed i.e. not modifiable. Tuples are defined inside parenthesis and values separated by comma.
- `>>> favorites=("Blue","Cricket","Gajar Ka Halwa")`
- `>>> student=(1,"Aman",97.5)`
- `>>> print(favorites)`
- **`('Blue', 'Cricket', 'Gajar Ka Halwa')`**
- `>>> print(student)`
- **`(1, 'Aman', 97.5)`**

# Dictionary Data Type

- ✓ Dictionary is another feature of Python. It is an unordered set of comma separated **key:value** pairs. Dictionary Items are defined in **Curly Brackets { }**
- ✓ Keys defined in Dictionary cannot be same i.e. no two keys can be same.
- ```
>>> student={'Roll':1,'Name':"Jagga",'Per':91.5}
```
- ```
>>>print(student)
```
- ```
>>> print(student['Per'])
```
- **91.5**
- ```
>>> val={1:100,2:300,4:900} # Key name can be string / numeric
```
- ```
>>> print(val[1])
```
- **100**
- **Dictionary is mutable. i.e. We can modify dictionary elements.**
- ```
>>>val[2]=1000
```
- ```
>>>print(val)           # {1: 100, 2: 1000, 4: 900}
```

Data type summary



Mutable and immutable types

- Python data object can be broadly categorized into two types – mutable and immutable types. In simple words changeable/modifiable and non-modifiable types.
- 1. **Immutable types:** are those that can never change their value in place. In python following types are immutable: **integers, float, Boolean, strings, tuples**
- **Sample Code:**

```
a = 10
```

```
b = a
```

```
c = 15
```

```
a = 20
```

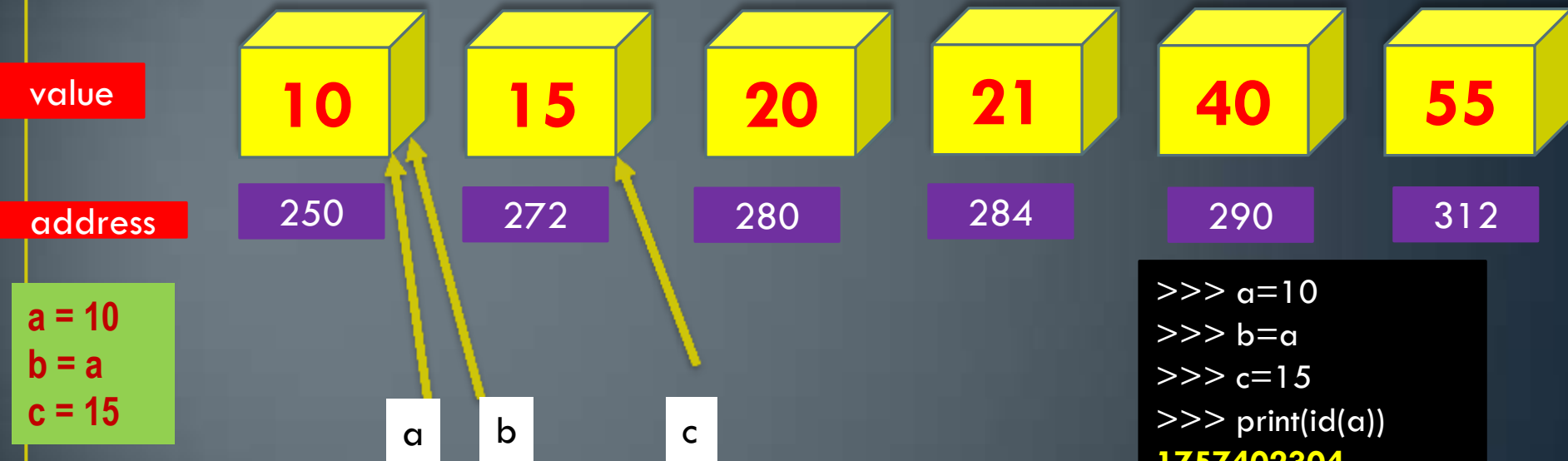
```
b = 40
```

```
c = b
```

From this code, you can say the value of integer a, b,c could be changed effortlessly, but this is not the case. Let us understand what was done behind the scene

immutable types

- Note: In python each value in memory is assigned a memory address. So each time a new variable is pointing to that value they will be assigned the same address and no new memory allocation. Let us understand the case.



Python provides **id()** function to get the memory address to which value /variable is referring

```
>>> a=10
>>> b=a
>>> c=15
>>> print(id(a))
1757402304
>>> print(id(b))
1757402304
>>> print(id(c))
1757402384
>>> print(id(10))
1757402304
```

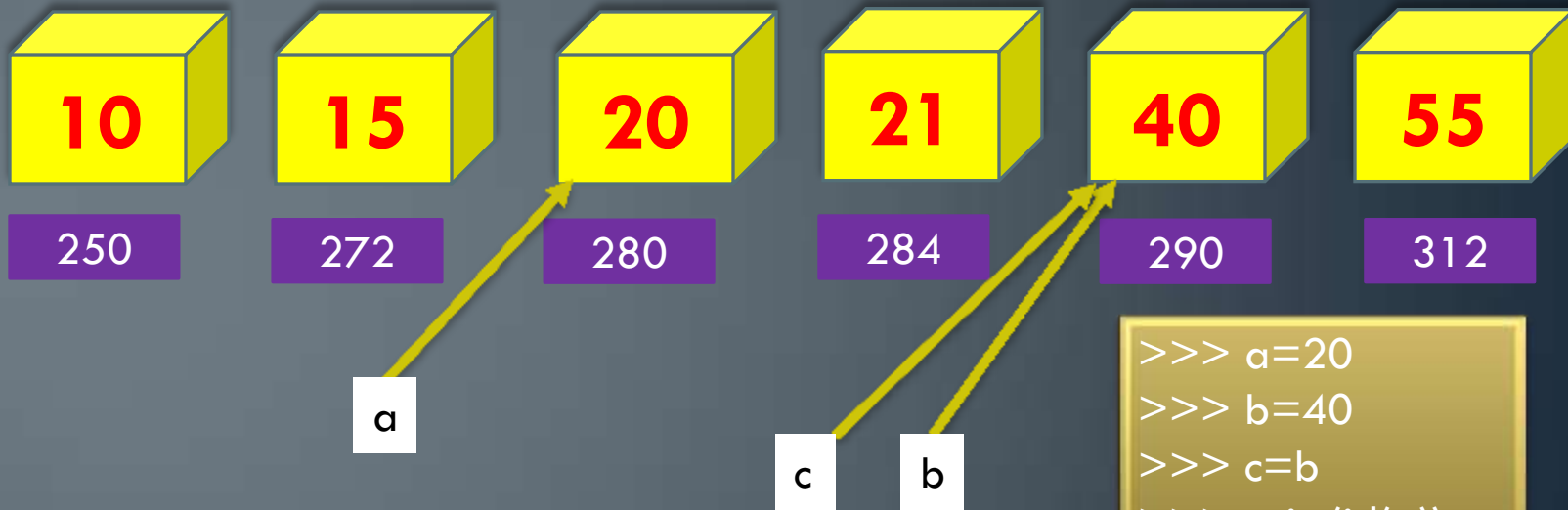
immutable types

Now let us understand the changes done to variable a, b, c

```
a = 20  
b = 40  
c = b
```

value

address



Python provides **id()** function to get the memory address to which value /variable is referring

```
>>> a=20  
>>> b=40  
>>> c=b  
>>> print(id(a))  
1757402464  
>>> print(id(b))  
1757402784  
>>> print(id(c))  
1757402784
```


immutable types

- From the previous code it is clear that variable names are stored references to a value-object. Each time we change the value the variable's reference memory address changes. So it will not store new value in same memory location that's why **Integer, float, Booleans, strings and tuples are immutable**.
- Variables (of certain type) are NOT LIKE storage containers i.e. with fixed memory address where value changes every time. Hence they are immutable

MUTABLE TYPE

- Mutable means in same memory address, new value can be stored as and when it is required. Python provides following mutable types:

1. Lists
2. Dictionaries
3. Sets

• Examples: (using List)

```
>>> employee=["E001","Rama","Sales",67000]
```

```
>>> print(id(employee))
```

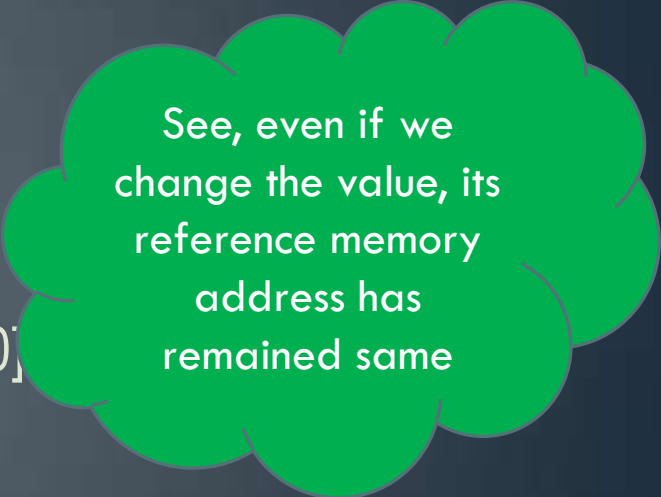
```
71593896
```

```
>>> employee[3]=75000
```

```
>>> print(id(employee))
```

```
71593896
```

```
>>>
```



See, even if we change the value, its reference memory address has remained same

VARIABLE INTERNALS

- Python is an object oriented language. So every thing in python is an object. An object is any identifiable entity that have some characteristics/properties and behavior. Like integer values are object – they hold whole numbers only(characteristics) and they support all arithmetic operations (behavior).
- Every python object has three key attributes associated with it:
 1. **type** of object
 2. **value** of an object
 3. **id** of an object

TYPE OF AN OBJECT

type of an object determines the operations that can be performed on the object. Built – in function `type()` returns the type of an object

Example:

```
>>> a=100
```

```
>>> type(a)
```

```
<class 'int'>
```

```
>>> type(100)
```

```
<class 'int'>
```

```
>>> name="Jaques"
```

```
>>> type(name)
```

```
<class 'str'>
```

Evaluating Logical expressions

- While evaluating logical expressions, Python follows these rules:
- (i) the precedence of logical operators is lower than arithmetic operators. For e.g.
10/5 or 5.0 + 50/10 will be evaluated as 5 or 10.0
- The precedence of logical operators among themselves are ***NOT , AND , OR***. So,
- (x and y or z and (not q)) will be evaluated as –
((x and y) or (z and (not q)))
- ***PYTHON USES SHORT-CIRCUIT CONCEPT WHILE EVALUATING LOGICAL EXPRESSION***

Type Casting

An explicit type conversion is user-defined conversion that forces an expression to be of specific type. The explicit type conversion is also known as Type Casting.

It is done using the syntax : **datatype_to_convert(expression)**

For example:

```
str='100' # String type
```

```
Num = int(str) # will convert the str to int type
```

Math Library

- `import math` # to include math library
- Important functions:

*ceil(), sqrt(), exp(), fabs(), floor(), log(), log10(),
pow(), sin(), cos(), tan()*