CONTENTS

(Learning Outcomes) ☺

FEATURES OF PYTHON:

- Python Character Set
- Tokens→(Keywords, Identifiers, Literals, Punctuators, Operators)
- Knowledge of Data Types and Operators
- Accepting input from the console
- Assignment statement
- Mutable/Immutable data types
- Introduction to Variable,
- Variable Internals and methods to manipulate it,
- Concept of L-value and R-value of a variable
- Operators and Types and their precedence:

  Binary and Unary Operators, Arithmetic Operators, Augmented Assignment Operators, Relational Operators, Logical Operators,
- Expressions

PYTHON CHARACTER SET

A set of valid characters recognized by python. Python uses the traditional ASCII character set. The latest version recognizes the Unicode character set. The ASCII character set is a subset of the Unicode character set.

- **Letters :- A-Z , a-z**
- **Digits:- 0-9**
- **Special symbols :- Special symbol available over keyboard**
- **White spaces:- blank space, tab, carriage return, new line, form feed**
- **Other characters:- Unicode**

TOKENS

Smallest individual unit in a program is called a token.

They are:

- **K**eywords
- **I**dentifiers
- **L**iterals
- **P**unctuators
- **O**perators

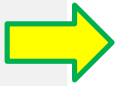
My funda of remembering Tokens: KILPO →it's just a shortcut given by me for remembrance ;)

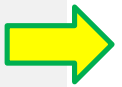
TOKEN→ LITERALS / values

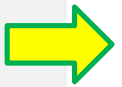
Literals are data items that have a fixed value.
Python allows several kind of literals:

Numeric Literals

- **Int (signed integers) →** Positive or Negative whole numbers with no decimal point Eg. 10, -96, 1234, 0
- **Floating point/Real values →** float represent real numbers and are written with a decimal point dividing the integer and fractional part. Eg 2.0,54.7,-12.0,-0.075, .4
- **Complex (complex numbers) →** a+b*j* , where a and b are floats and J(or j) represents $\sqrt{-1}$, which is an imaginary number. a is the real part and b is the imaginary part of the number.







```
>>> p=10
>>> p
10
>>> q=7.5
>>> q
7.5
>>> r=4+9j
>>> r
(4+9j)
```

TOKEN→ LITERALS contd..

Boolean Literals

True and False are the only two Boolean values

Special Literal *None*

None literal represents absence of a value. That is why nothing is printed for the value of b.

String Literal

String literal is a sequence of characters surrounded by quotes (single or double or triple)

String literals can be written in either single quote ‘ ’ or double quote “ ”

```
>>> a=True
>>> a
True
>>> b=None
>>> b
>>> s="India"
>>> s
'India'
```

Python Escape Characters

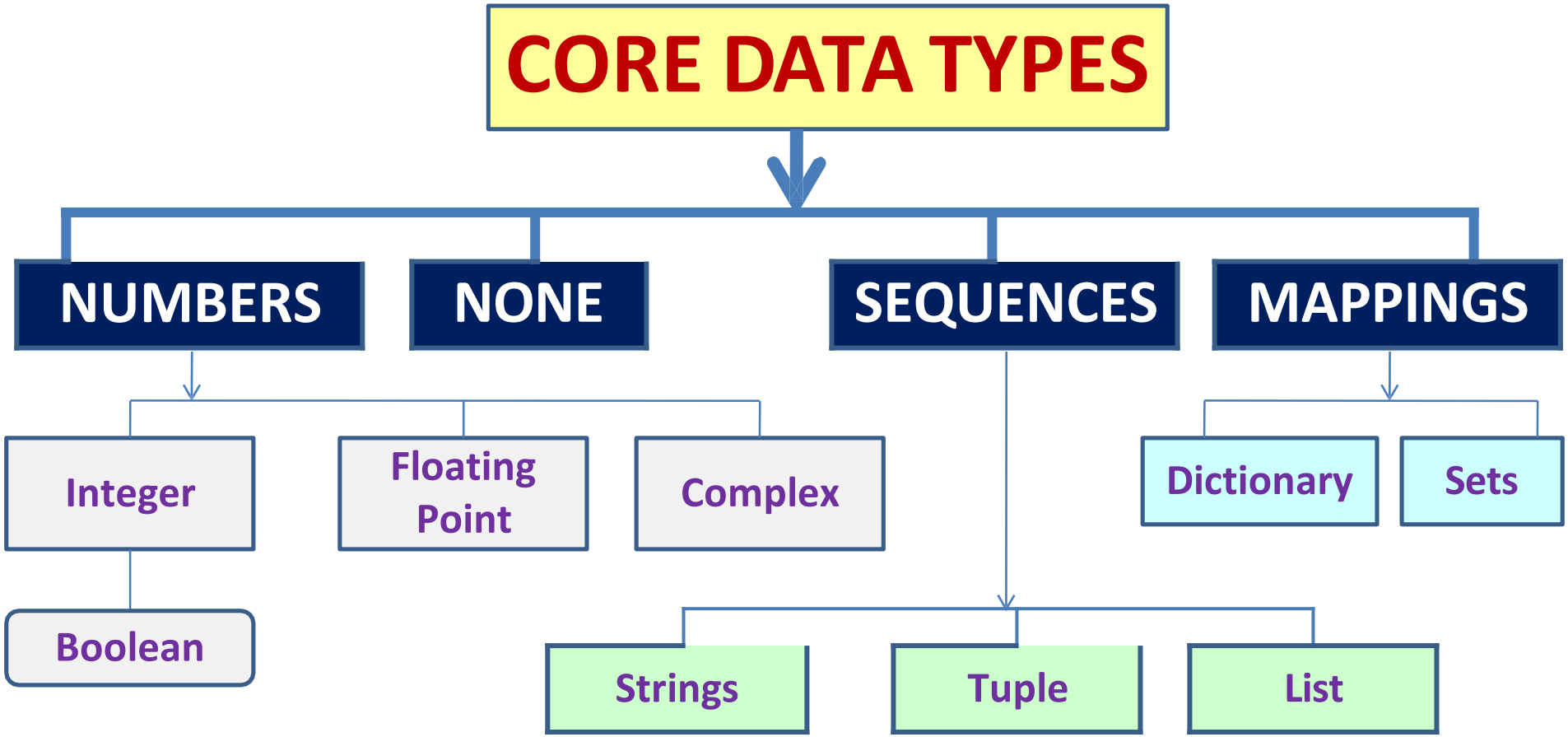
Code	Result/Output	Description
\'	Single Quote	Add single quote with in a String
\\	Backslash	Insert single Back Slash
\n	New Line	\n takes the cursor to first position of a new line
\r	Carriage Return	\r takes the cursor to the first position of the same line
\t	Tab	\t add spaces of 8 characters
\b	Backspace	\b takes the cursor one position backward
\f	Form Feed	Form Feed is page breaking ASCII control character
\ooo	Octal value	Octal value
\xhh	Hex value	Hex value

DATA TYPES

IN PYTHON

DATA TYPES

The kind of data for handling the data used in a program code is the ***data type***.



MUTABLE AND IMMUTABLE TYPES

- **Mutable types:**

  The mutable types are those whose values can be changed in place.

  Only three types are mutable in Python.

  These are: **Lists, dictionaries and sets.**

- **Immutable types:**

  The immutable types are those that can never change their value in place.

  In python , the following are the immutable types:

Integers, floating point numbers, booleans, strings, tuples

DATA TYPE: Number in Python

It is used to store numeric values.

Python has three numeric types:

1. Integers

2. Floating point numbers

3. Complex numbers.

DATA TYPE: NUMBER → Integer

Integer or int are positive or negative numbers with no decimal point. Integers in Python are of unlimited size.

```
>>> a=10
>>> b=-20
>>> a*b
-200
>>> print(a)
10
>>> print(b)
-20
>>> print(a*b)
-200
```

Type Conversion to Integer , int( ) function

int() function converts any data type to integer.

```
>>> s="105"
>>> a=int(s)
>>> a
105
>>> f=-25.7
>>> int(f)
-25
```

DATA TYPE: NUMBER → Float

Float are positive or negative real numbers with decimal point. Some Floating point number examples:

4.5, 2.0, -38.9,-18.0

```
>>> p=92.5
>>> p
92.5
>>> q=-44.7
>>> q
-44.7
>>> p*q
-4134.75
>>> r=p*q
>>> r
-4134.75
```

Type Conversion to Float , float( ) function

float() function converts any data type to floating point numbers.

```
>>> s="468"
>>> a=float(s)
>>> a
468.0
>>> a=float(-24)
>>> a
-24.0
```

DATA TYPE: Complex Number

- Complex numbers are **combination of a real and imaginary part.**
- Complex numbers are in the form of **X+Yj**

  , where **X is a real part** and **Y is imaginary part.** We know, in mathematics, **j stands for $\sqrt{-1}$**
- Using **complex() function** , we can also create complex numbers.

```
>>> a=12+5j
>>> a
(12+5j)
>>> b=34-81j
>>> b
(34-81j)
>>> c=-8j
>>> c
(-0-8j)
>>> a+b
(46-76j)
>>> print(b-c)
(34-73j)
```

```
>>> complex(5,-27)
(5-27j)
>>> x=complex(-45,-23)
>>> x
(-45-23j)
>>> y=complex(0,4)
>>> y
4j
>>> z=complex(32)
>>> z
(32+0j)
>>> y+x
(-45-19j)
```

DATA TYPE: Boolean

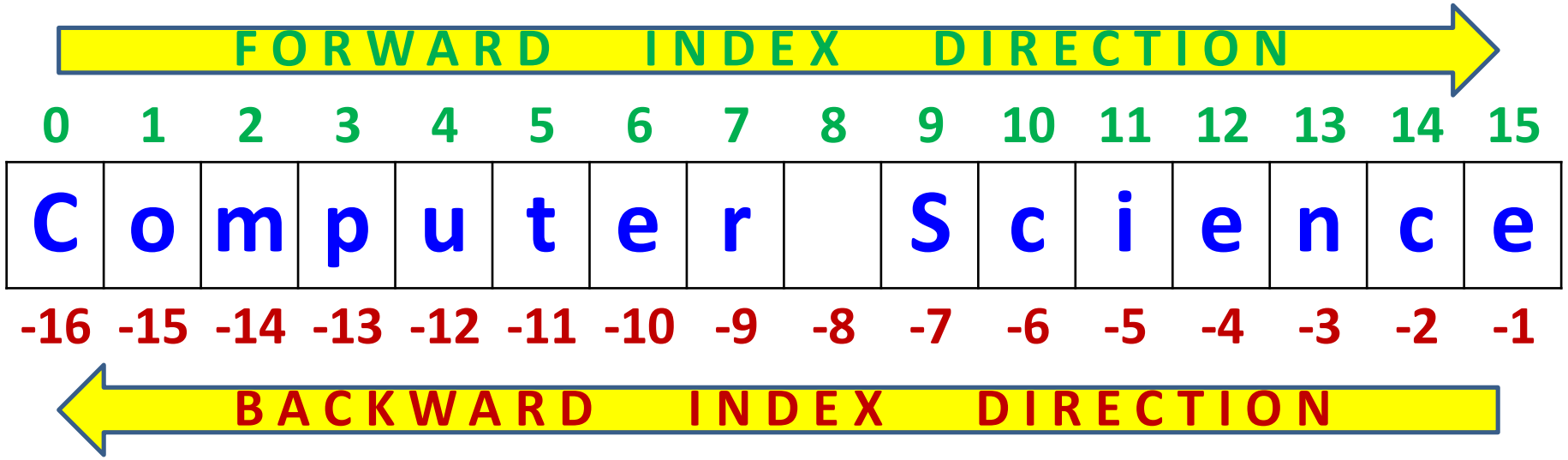
- **Boolean takes any two values, either** **True** **or** **False**
- Every **integer** number, **float** number or **complex** number whether **negative or positive** takes a Boolean value **True.**
- Value **0 , 0.0 , 0+0j are considered** **False** in Boolean.

```
>>> bool(1)
True
>>> bool(5)
True
>>> bool(0.7)
True
>>> bool(-234)
True
>>> bool(-74.58)
True
```

```
>>> bool(4+17j)
True
>>> bool(0-12j)
True
>>> bool(3+0j)
True
>>> bool(0+0j)
False
>>> bool(0)
False
>>> bool(0.0)
False
```

DATA TYPE: STRING in Python

A string is a sequence of characters. In python, we can create string using single(') or double quotes("). Both are same in python.



NOTE:-

The index (also called subscript) is the numbered position of a letter in the string. In python, indices begin from

0,1,2,.... upto (length-1) in the *forward direction* and

-1,-2,-3,..... (-length) in the *backward direction*.

where length is the length of the string.

DATA TYPE: working with STRING examples

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	o	m	p	u	t	e	r		S	c	i	e	n	c	e	!	!	!
-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> s="Computer Science!!!"
>>> s
'Computer Science!!!'
>>> s[3]
'p'
>>> s[-3]
'!'
>>> s[-6]
'n'
>>> s[3:]
'puter Science!!!'
>>> s[:12]
'Computer Sci'
>>> s[-8:]
'ience!!!'
>>> s[:-13]
'Comput'
```

```
>>> s[5:16]
'ter Science'
>>> s[7:11]
'r Sc'
>>> s[-4:-8]
''
>>> s[-8:-2]
'ience!'
>>> s[-2:6]
''
>>> s[6:-2]
'er Science!'
>>> print("I Love "+s)
I Love Computer Science!!!
>>> s*2
'Computer Science!!!Computer Science!!!'
```

DATA TYPE: String contd..

Python allows to have two string types:

1) Single line strings (Basic strings)

Text enclosed in single quote 'India' or in double quotes "Bharat" are normally single line strings i.e. they must terminate in one line.

```
>>> s='India'
>>> s
'India'
>>> s="Bharat"
>>> s
'Bharat'
```

2) Multiline strings

Text spread across multiple lines as one single string.
Two ways of treating multiline strings:-

(i) By adding a backslash at the end of normal single quote/double quote strings.

```
>>> s="Welcome\
to the\
World of\
wonders"
>>> s
'Welcometo theWorld ofwonders'
```

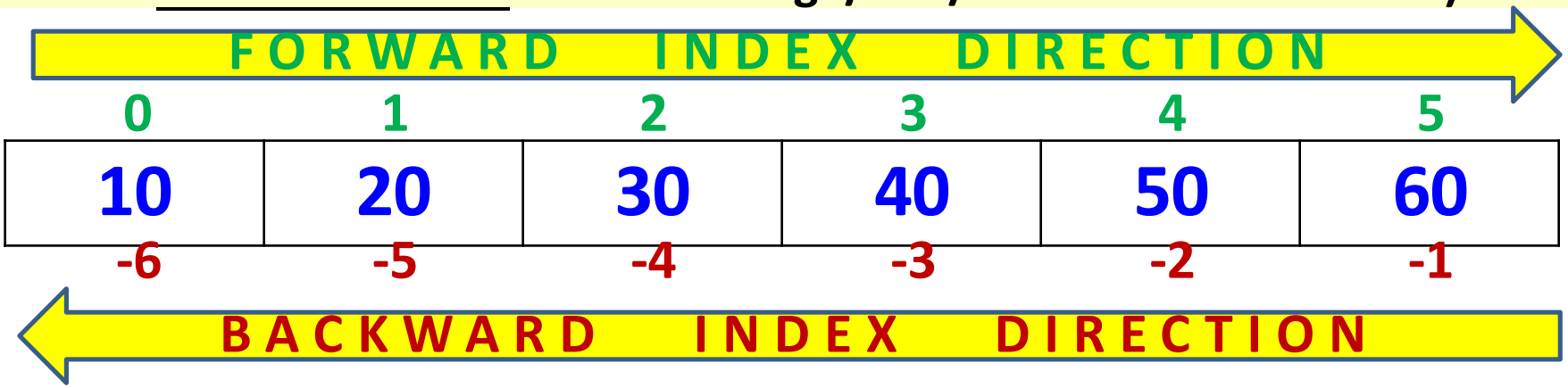
(ii) By typing the text in triple quotation or apostrophe mark (No backslash needed at the end of line)

```
>>> Str='''Welcome
to the
world of wonders.'''
>>> Str
'Welcome\nto the\nworld of wonders.'
>>> A="""Welcome
to the
world of wonders."""
>>> A
'Welcome\nto the\nworld of wonders.'
```

DATA TYPE: LIST in Python

A list in Python represents a list of comma-separated values of any data type between square brackets e.g. following are some lists:

LIST is mutable- one can change/add/delete a list's element)



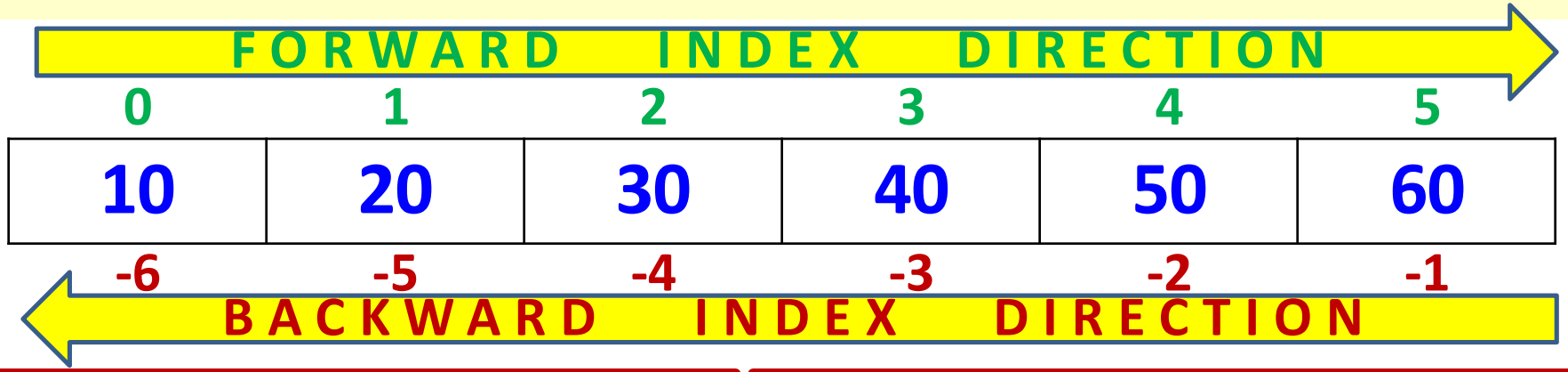

```
>>> [1,2,3,4,5]
[1, 2, 3, 4, 5]
>>> ['a','g','n','i']
['a', 'g', 'n', 'i']
>>> [10,20,12.5,'P',64]
[10, 20, 12.5, 'P', 64]
>>> L=[10,20,30,40,50,60]
>>> L
[10, 20, 30, 40, 50, 60]
```

```
>>> L[0]
10
>>> L[-1]
60
>>> L[3]=-250
>>> L[-5]=95
>>> L
[10, 95, 30, -250, 50, 60]
```

DATA TYPE: TUPLE in Python

Tuples are represented as list of comma-separated values of any data type within parentheses. Some examples of tuples:

Tuple is immutable - one cannot change the values of a tuple.

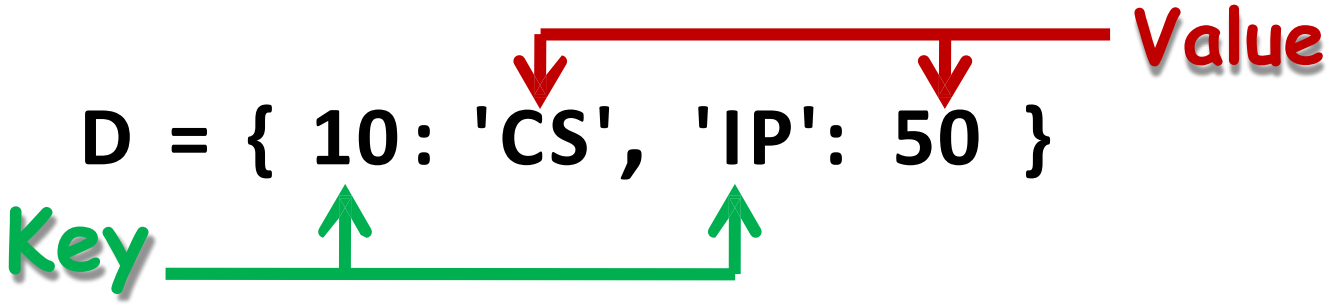



```
>>> (10,20,30,45,58)
(10, 20, 30, 45, 58)
>>> ('A','g','n','i')
('A', 'g', 'n', 'i')
>>> T=(10,-20,-44,15,'R')
>>> T
(10, -20, -44, 15, 'R')
```

```
>>> T[0]
10
>>> T[-2]
15
>>> T[1]=33
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in
<module>
    T[1]=33
TypeError: 'tuple' object does not
support item assignment
```

DATA TYPE: DICTIONARY in Python

Dictionary data type is an **unordered set of comma-separated key: value pairs, within { }**, with the requirement that within a dictionary, no two keys can be the same (i.e. there are unique keys within a dictionary). Some examples of Dictionary:




```
>>> D={1:'Jan',2:31}
>>> D
{1: 'Jan', 2: 31}
>>> D.keys()
dict_keys([1, 2])
>>> D.values()
dict_values(['Jan', 31])
>>> print(D)
{1: 'Jan', 2: 31}
>>> D[2]
31
>>> D[1]
'Jan'
```

```
>>> Dict={10:'CS','IP':50}
>>> Dict[10]
'CS'
>>> Dict['IP']
50
>>> Dict
{10: 'CS', 'IP': 50}
>>> Dict.keys()
dict_keys([10, 'IP'])
>>> Dict.values()
dict_values(['CS', 50])
```

WORKING WITH VARIABLES

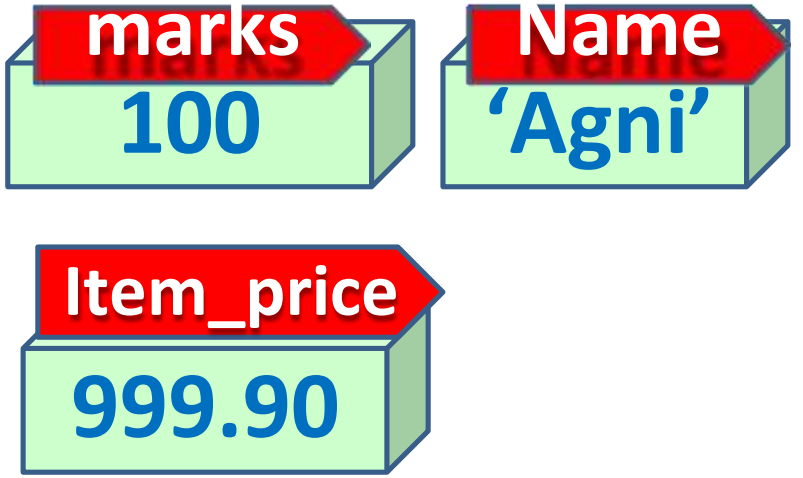
- CONCEPT OF VARIABLE
  - LVALUE AND RVALUE,
- INPUT AND OUTPUT FUNCTIONS,
- VARIABLE INTERNALS,
- ASSIGNING SAME/ MULTIPLE VALUE(S) TO MULTIPLE VARIABLES,
- DYNAMIC TYPING AND DYNAMIC TYPING ERROR

CONCEPT OF VARIABLES

- **Variable is a name *(Named Labels)* given to a memory location that refers to a value and whose values can be used and processed during program run.**
- Python variables are created by assigning value of desired data type to them. The assignment operator = is used to assign the values from RHS to LHS variable. Eg. *Assign integer value to create integer variable, string value to a variable to create string variable.*
- Python is a type infer language that means you don't need to specify the datatype of variable. Python automatically get variable datatype depending upon the value assigned to the variable.

```
>>> marks=100
>>> Name='Agni'
>>> Item_price=999.90
```

NOTE:
Here, ***marks, Name*** *and* ***Item_price*** are the **named labels (variables)** with datatype inferred as ***Integer, String*** *and* ***Float*** for the values ***100, 'Agni', 999.90*** respectively.



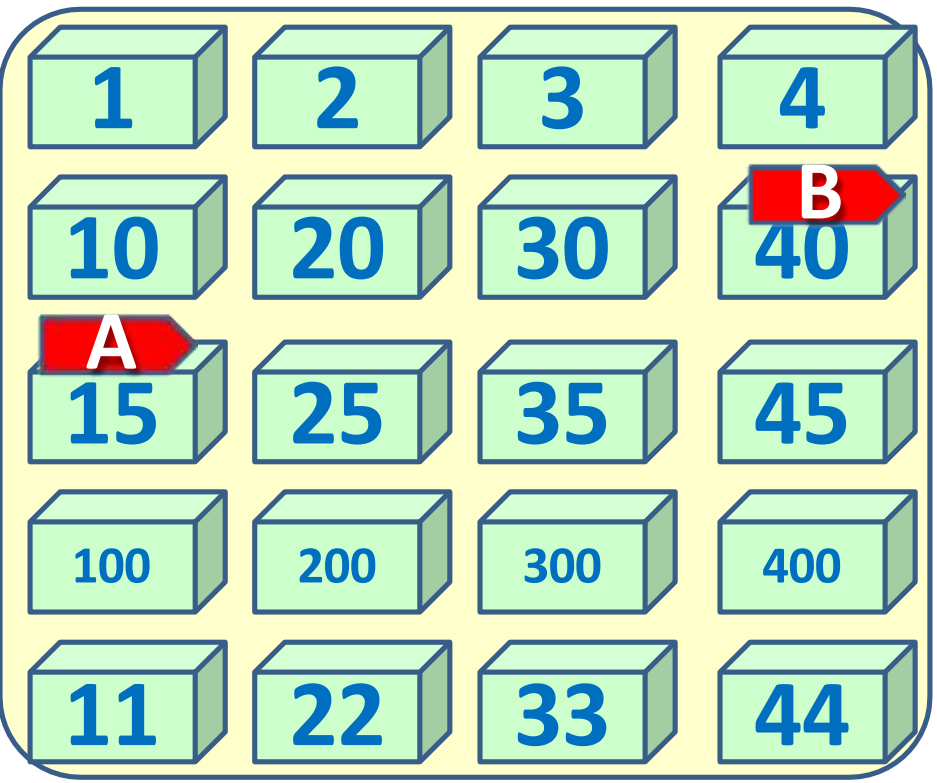
MEMORY LOCATIONS: Imagine a Table top with different boxes having some values stored in them. The variables which you will create are the tags *(Like price tags on items in a mall)* , which you will place on the top of the boxes.

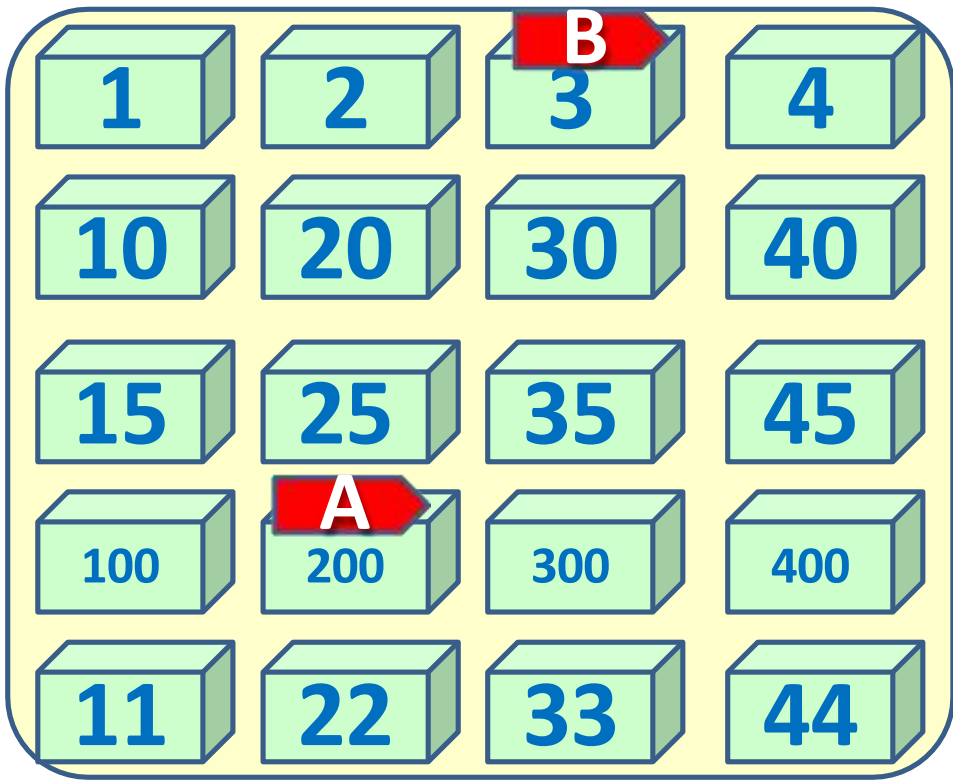
```
>>> A=15
>>> B=40
>>> A
15
>>> B
40
```

These tags(variables) will represent (or point) those boxes with the value assigned to the variables. Unlike other programming languages,

VARIABLES ARE NOT SORAGE CONTAINERS in PYTHON

```
>>> A=200
>>> B=3
>>> A
200
>>> B
3
```





ASSIGNING VALUES TO VARIABLES

- A Variable is not created until some value is assigned to it.

```
>>> a
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a=10
>>> a
10
```

ASSIGNING SAME VALUE TO MULTIPLE VARIABLES

```
>>> x=y=z=10
>>> print(x,y,z)
10 10 10
```

ASSIGNING DIFFERENT VALUES TO MULTIPLE VARIABLES

```
>>> x,y,z=33,78,45
>>> print(x,y,z)
33 78 45
```

ASSIGNING VALUES TO VARIABLES THROUGH AN EXPRESSION.

```
>>> A=11
>>> B=20.4
>>> SUM=A+B
>>> SUM
31.4
```

ASSIGNING None VALUE (nothing) TO A VARIABLE

```
>>> sales=None
>>> sales
>>>
```

SWAPPING VARIABLES' VALUES

```
>>> x,y=2,5
>>> print(x,y)
2 5
>>> x,y=y,x
>>> print(x,y)
5 2
```

L-VALUE and R-VALUE OF A VARIABLE

Lvalue: Expressions which can be given on the **LHS(Left Hand Side)** of an assignment.

Rvalue: Expressions which can come on the **RHS(Right hand side)** of an assignment.

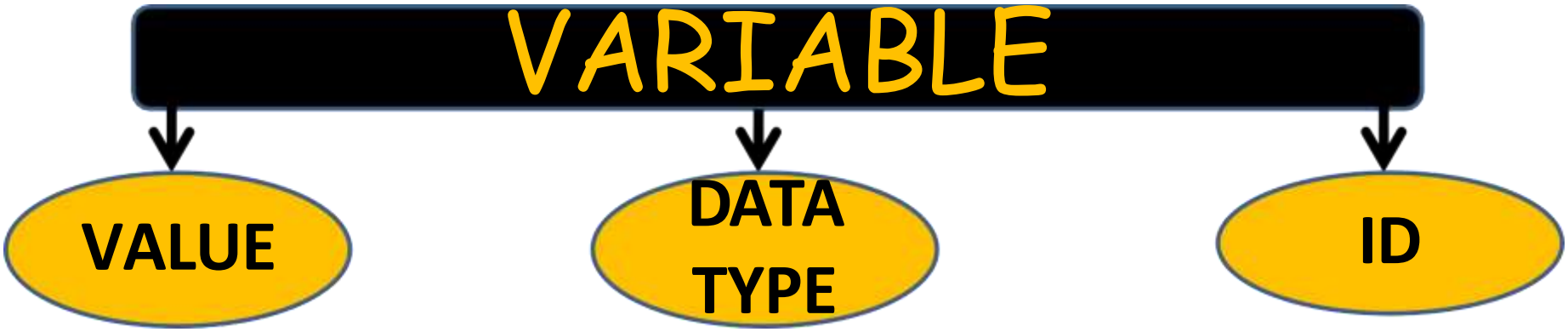
```
>>> x=1
>>> y=4
>>> x+y=z
SyntaxError: cannot assign to operator
>>> z=x+y
>>> print(z)
5
```

NOTE: In Python, assigning a value to a variable means, variable's label is referring to that value

```
>>> L=8
>>> B=3
>>> Area=L*B
>>> Area
24
>>> L*B=Area
SyntaxError: cannot assign to operator
```

We cannot assign a variable's *(Area)* value to an expression *(L*B)*

VARIABLE INTERNALS

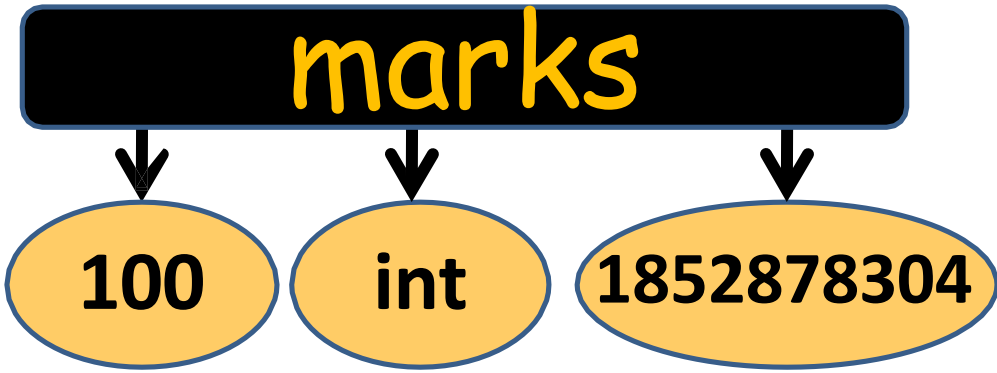


id of an object (Identity)

The id of an object is generally the **memory location of the object**. Although id is implementation dependent but in most implementations it returns the memory location of the object. **id()** is a built-in function that returns the id of an object.

Consider, the following code

```
>>> marks=100
>>> marks
100
>>> type(marks)
<class 'int'>
>>> id(marks)
1852878304
```



More on VARIABLE INTERNALS with example

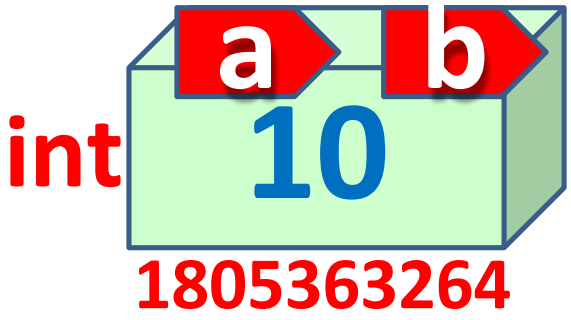
```
>>> a=10
>>> a
10
>>> type(a)
<class 'int'>
>>> id(a)
1805363264
```

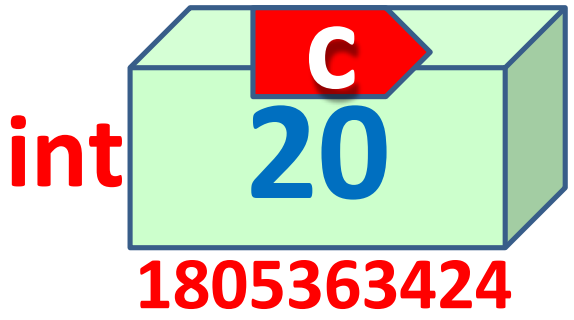
```
>>> b=10
>>> b
10
>>> type(b)
<class 'int'>
>>> id(b)
1805363264
```

```
>>> c=20
>>> c
20
>>> type(c)
<class 'int'>
>>> id(c)
1805363424
```

Here,

- a, b, c are named labels/ variable names /object names
- 10 and 20 are the values
- int is the data type or <class type> of the variables /objects
- 1805363264 and 1805363424 are the id of the objects






```
>>> a==b
True
>>> a==c
False
```

```
>>> type(a)==type(b)
True
>>> type(a)==type(c)
True
```

```
>>> id(a)==id(b)
True
>>> id(a)==id(c)
False
```

INPUT through input() function and OUTPUT through print() statement

- To take input from the user, input() function is used. input() function takes values in string data type.

Syntax of input Function

input()

- To print/display the contents on the console(monitor screen), print() function is used.

Syntax of print Function

print(expression/variable)

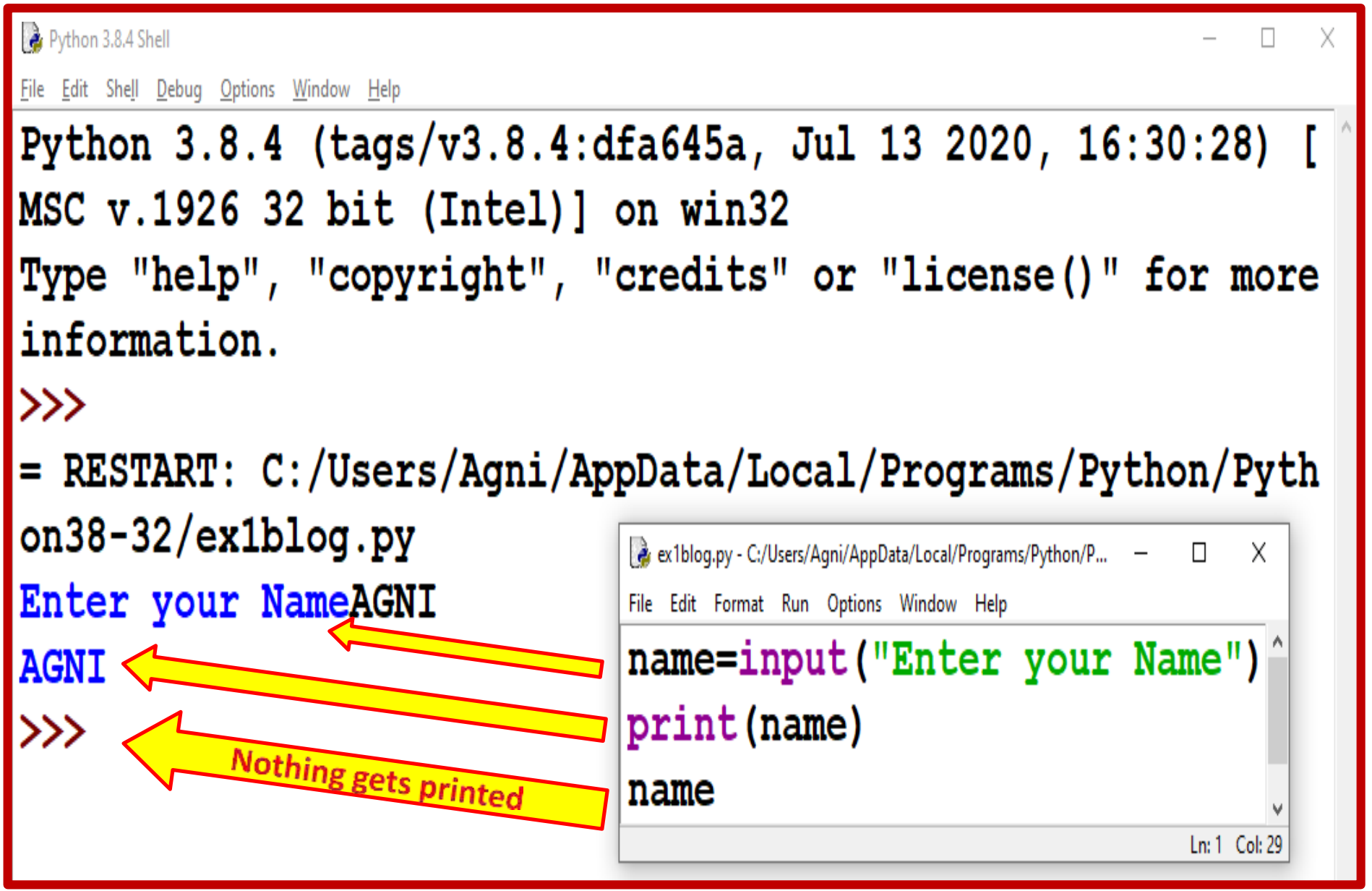
```
>>> name=input("Enter your Name")
Enter your NameAGNI
>>> name
'AGNI'
>>> print(name)
AGNI
```

The cursor / prompts the user to give some input

NOTE:

Interactive mode supports displaying of data , both with print() function and also directly writing the data (contents) on the Python shell prompt >>>

NOTE: But, to display in Script mode , print() function is required to display the data (contents), otherwise nothing is printed



More on INPUT through input() function and OUTPUT through print() statement.....

```
>>> item=input('Enter the item name')
Enter the item nameChocolate
>>> item
'Chocolate'
>>> type(item)
<class 'str'>
>>> price=input('Enter the item price')
Enter the item price100
>>> price
'100'
>>> type(price)
<class 'str'>
>>> price=int(input('Enter the price'))
Enter the price100
>>> price
100
>>> type(price)
<class 'int'>
```

input () takes string value by default.

But price should be of numeric value

#We can use int() or float () function with the input function

Some more on →input() and print()

```
>>> input()
Chota Bheem
'Chota Bheem'
```

```
>>> name=input()
Chota Bheem
>>> name
'Chota Bheem'
```

```
>>> name=input("Enter a cartoon character")
Enter a cartoon characterChota Bheem
>>> name
'Chota Bheem'
```

sep and **end** parameters of print() function.

sep is the separator parameter which separates the strings given in print function.
The default separator (in the absence of sep) is a space ' '.
end parameter denotes the end character to be given at the last of the print() functions data/content.

```
>>> print('Computer','Science')
Computer Science
>>> print('Computer','Science',sep='&')
Computer&Science
>>> print('Computer','Science',sep='-',end='!')
Computer-Science!
>>> print('Python language is easy',sep='-',end='.')
Python language is easy.
>>> print('Python','language','is','easy',sep='-',end='.')
Python-language-is-easy.
```

DYNAMIC TYPING

A Variable pointing to a value of a certain type, can be made to point to a value/object of different type.

This is called Dynamic typing.

```
>>> a=100
>>> print(a)
100
>>> a='sky'
>>> a
'sky'
```

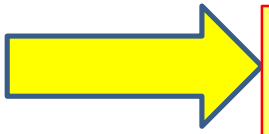
NOTE:
Here, variable **a** changes its data type from 'int' to 'string' dynamically (i.e. during run time).

DYNAMIC TYPING error

```
>>> a=100
>>> print(a-2)
98
>>> a='sky'
>>> print(a-2)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    print(a-2)
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

CHAINED COMPARISON OPERATOR

```
>>> 10<20<35
True
>>> 10<20 and 20<35
True
```

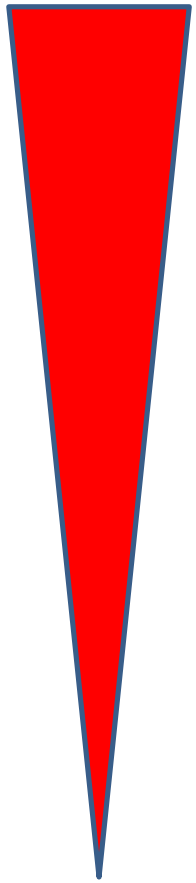


Both the statements are equivalent. Hence, the output is same.

OPERATOR PRECEDENCE

Operator	Description
**	Exponentiation (raised to the power)
~+-	Complement unary plus and minus(method names for last two are +@ and -@)
*/%//	Multiply, divide, modulo and floor division
+-	Addition and Subtraction
>> <<	Right and left bitwise shift
&	Bitwise ‘AND‘
^|	Bitwise exclusive ‘OR' and regular ‘OR'
<= <> >=	Comparison operators
< > == !=	Equality operators
= %= /= //= += ‘= “=	Assignment operators
is , is not	Identity operators
in , not in	Membership operators
not , or , and	Logical operators

Highest Priority



Lowest Priority

OPERATOR ASSOCIATIVITY

- Left-to-right associativity for all the operators except exponentiation ** operator
- For eg. multiplication(*),division (/) and floor division operator(//) have the same precedence,
- In same expression→ the operator on the left is evaluated first and then the operator on its right and so on.

```
>>> 4*85/50//3
2.0
```

Left to Right Associativity

Same as

```
>>> ((4*85)/50)//3)
2.0
```

```
>>> 4*85
340
>>> 340/50
6.8
>>> 6.8//3
2.0
```

Here, * / // have the same precedence level. So, by default left to right associativity while execution.

Some more examples on OPERATOR ASSOCIATIVITY

```
>>> 33+8/2
37.0
```

Not the same

```
>>> ((33+8)/2)
20.5
```

Same as

```
>>> (33+(8/2))
37.0
```

NOTE:

Here, in the expression / (Division operator) has the higher priority the + (Addition operator) in the precedence table.

```
>>> 2**3**4
2417851639229258349412352
```

Not the same

```
>>> ((2**3)**4)
4096
```

Same as

```
>>> (2**(3**4))
2417851639229258349412352
```

EXCEPTIONAL CASE:

If an expression contains **, then this exponentiation operator has right to left associativity while execution.

EXPRESSIONS

- TYPES OF EXPRESSIONS
- DATA TYPE CONVERSIONS
  - → IMPLICIT TYPE CONVERSION
  - → EXPLICIT TYPE CONVERSION

Expression= Atom + Operators

- An atom is something that has a value. Identifiers, literals, strings, lists, tuples, sets, dictionaries etc.
- An atom is something that has some value.

EXPRESSION: It is referred as a valid combination of operators and atoms. An expression is composed of one or more operations.

Types of Expressions

(i) Arithmetic expressions

(ii) Relational expressions

(iii) Logical expressions

(iv) String expressions

Two string operators + and *

```
>>> 5+8
13
>>> 4<7
True
>>> 'a' or 'b'
'a'
>>> "Mother"+"India"
'MotherIndia'
>>> "India"*3
'IndiaIndiaIndia'
```

DATA TYPE CONVERSIONS

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called **type conversion.**

Python has two kinds of type conversion.

i) Implicit Type Conversion

ii) Explicit Type Conversion

IMPLICIT TYPE CONVERSION

In implicit type conversion, Python **automatically converts one data type to another data type**. This process doesn't need any user involvement.

Eg.

```
>>> i=10
>>> f=2.34
>>> sum=i+f
>>> print("Datatype of i =",type(i))
Datatype of i = <class 'int'>
>>> print("Datatype of f =",type(f))
Datatype of f = <class 'float'>
>>> sum
12.34
>>> print("Datatype of sum =",type(sum))
Datatype of sum = <class 'float'>
```

EXPLICIT TYPE CONVERSION

In Explicit Type Conversion, **user converts the data type of an object to required data type.** We use the predefined functions like int(),float(),str() etc.

Eg.

```
>>> i=20
>>> s="45"
>>> print("Data type of i =",type(i))
Data type of i = <class 'int'>
>>> print("Data type of s BEFORE type casting =",type(s))
Data type of s BEFORE type casting = <class 'str'>
>>> s=int(s)
>>> print("Data type of s AFTER type casting =",type(s))
Data type of s AFTER type casting = <class 'int'>
>>> sum=i+s
>>> sum
65
>>> print("Data type of sum =",type(sum))
Data type of sum = <class 'int'>
```

Changing Data type of S from <str> type to <int> type